

# FORMAL VERIFICATION AND AUTOMATED MASKING OF CRYPTOGRAPHIC HARDWARE



DISSERTATION

---

*zur Erlangung des Grades eines Doktor-Ingenieurs  
der Fakultät für Informatik  
an der Ruhr-Universität Bochum*

---

*by David Knichel  
Bochum, May 2023*



To my parents, Sigrid and Jürgen, for unconditional love and endless support.

And to Hannah, the love of my life.



---

David Knichel  
Place of birth: Essen, Germany  
Author's contact information:  
`david.knichel@rub.de`

Thesis Advisor: **Prof. Dr. Amir Moradi**  
Ruhr University Bochum, Germany  
Secondary Referee: **Prof. Dr. François-Xavier Standaert**  
Université catholique de Louvain, Belgium  
Tertiary Referee: **Prof. Dr. Sebastian Faust**  
TU Darmstadt, Germany  
Thesis submitted: May 22, 2023  
Thesis defense: June 19, 2023  
Last revision: July 23, 2023



# Abstract

Initially used in niche military applications, cryptography has evolved to be the indispensable foundation for guaranteeing user privacy and safety in today's data-driven and universally connected world. Technological mega trends like Artificial Intelligence (AI) and the Internet of Things (IoT) vastly accelerate the collection and processing of unimaginable amounts of personal data whose confidentiality, integrity and authenticity must be protected. Nowadays, cryptography is not only a mere extension to a system, but it even lies at the heart of many applications' functionality, for example, in the context of decentralized networks.

As a result, cryptographic algorithms are widely implemented in both soft- and hardware on a heterogeneous spectrum of devices ranging from payment cards, over key tokens and automotive platforms, to modern smartphones. As cryptographic algorithms often have to be realized within a resource-constrained context while guaranteeing high throughput and low latency, many chip vendors have started integrating cryptography as a dedicated, on-chip hardware module. Such accelerators are highly attractive as they drastically improve performance compared to sole software implementations.

While all common cryptographic algorithms are considered secure from a mere cryptanalytic viewpoint, i.e., when an adversary is limited to choosing and observing in- and outputs of the algorithm, and the execution itself is considered a black box, many works have shown that this theoretical restriction of adversarial behavior is not accurate in various realistic scenarios. In many cases, an attacker can access additional information during the execution of a cryptographic implementation, weakening the assumption of a black-box model. By passively observing the physical characteristics of a device, like the timing, the power consumption, or the electromagnetic emanation during the execution of a cryptographic algorithm, an adversary may obtain confidential information that is currently processed. We refer to attacks that leverage such unintentional relations as Side-Channel Analysis (SCA) attacks. Since their discovery, SCA attacks have proven to be highly effective for the purpose of recovering key material and breaking the security of many real-world and widespread devices.

This is why intensive focus has been laid on finding strong countermeasures. A prominent and well-studied protection against SCA is offered by the *masking* countermeasure, which, since its introduction in the context of SCA two decades ago, has received the lion's share of attention due to its sound theoretical foundation rooted in the concept of secret sharing. Here, a randomized split of the information in the processed data aims to move the successful recovery of secret keys into the practically impossible.

Achieving thorough masking in hardware is a tedious and delicate task, even for experts that can look back on many years of experience in building SCA-resistant hardware implementations. Due to the vast variety of leakage-driving physical effects within digital hardware circuits, minor errors in the design can render an implementation entirely insecure in practice. Starting from a process that mainly involved masking based on heuristics and experimental evaluation of the resulting hardware implementation through standard leakage assessment methodologies,

---

part of the research efforts has been devoted to deriving more systematic masking approaches and evaluating designs early on in the design process. The establishment of formal adversary models, which realistically, but abstractly, model an SCA adversary's capabilities and define what information leakage can be extracted from a circuit, was a key enabler for deriving both systematic masking that can even be entirely automated, and security verification based on the netlist of the circuit. Due to its high abstraction, the robust  $d$ -probing model is a well studied and highly convenient SCA adversary model in the hardware context.

While generally aiming to drive advances with respect to ensuring strong protection against power-related SCA attacks, this thesis particularly presents contributions in three categories. We underline the ongoing relevance of SCA attacks – even for complex, modularized systems – by showing how combining software exploits and SCA can pose a potent threat to user privacy in widely spread consumer devices. We furthermore introduce new techniques to realize formal verification, i.e., the attestation of security and composability of masked hardware designs with respect to the formal robust  $d$ -probing adversary model. Last but not least, we introduce several novel schemes and methodologies for systematic and automated masking. With these, we hand an engineer a toolbox for automatically transforming any digital circuit into a circuit thoroughly protected against SCA attacks while offering high flexibility to find a suitable trade-off between security level, latency and area footprint of the resulting design.

**Keywords.**

Cryptography, Hardware Security, Side-Channel Attacks, Formal SCA Adversary Models, Masking, Composability, Formal Verification, Automation, iPhone Security



# Kurzfassung

## Formale Überprüfung und automatisierte Maskierung von Kryptografischer Hardware

Ursprünglich in militärischen Nischenanwendungen eingesetzt, hat sich die Kryptografie in der heutigen datengesteuerten und global vernetzten Welt zu einer unverzichtbaren Grundlage für die Gewährleistung der Privatsphäre und Sicherheit der Nutzer entwickelt. Technologische Megatrends wie Künstliche Intelligenz (KI) und das Internet der Dinge (IoT) beschleunigen die Erfassung und Verarbeitung unvorstellbarer Mengen personenbezogener Daten enorm. Daher ist es unerlässlich, die Vertraulichkeit, Integrität und Authentizität dieser Daten zu schützen. Kryptografie ist heute nicht mehr nur eine Erweiterung eines Systems oder einer Anwendung, sondern das Herzstück vieler Anwendungsfunktionen, z. B. im Zusammenhang mit verteilten Netzwerken.

Kryptografische Algorithmen sind sowohl in Soft- als auch in Hardware auf einem heterogenen Spektrum unterschiedlicher Geräte implementiert, welches von Zahlungskarten, über Schlüssel-Token und Automobilplattformen, bis hin zu modernen Smartphones reicht. Da kryptografische Algorithmen oft in einem ressourcenbeschränkten Kontext realisiert werden müssen und gleichzeitig ein hoher Durchsatz und eine niedrige Latenzzeit garantiert werden müssen, haben viele Chip-Hersteller damit begonnen, die Kryptografie als dediziertes On-Chip Hardwaremodul zu integrieren. Solche Hardwarebeschleuniger sind äußerst attraktiv, da sie die Effizienz im Vergleich zu reinen Softwareimplementierungen drastisch verbessern.

Während alle gängigen kryptografischen Algorithmen unter rein kryptanalytischen Gesichtspunkten als sicher gelten, d.h. wenn ein Angreifer nur die Ein- und Ausgaben des Algorithmus auswählen und beobachten kann und die Ausführung selbst als Black Box betrachtet wird, haben viele Arbeiten gezeigt, dass diese theoretische Einschränkung des gegnerischen Verhaltens in verschiedenen realistischen Szenarien nicht zutreffend ist. In vielen Fällen ist ein Angreifer in der Lage, während der Ausführung einer kryptografischen Implementierung auf zusätzliche Informationen zuzugreifen, wodurch die Annahme eines Black-Box Modells unzulässig wird. Durch die passive Beobachtung der physikalischen Eigenschaften eines Geräts, wie z. B. des Timings, des Stromverbrauchs oder der elektromagnetischen Abstrahlung während der Ausführung eines kryptografischen Algorithmus, kann ein Angreifer vertrauliche Daten beziehen, die gerade verarbeitet werden. Wir bezeichnen Angriffe, die solche unbeabsichtigten Zusammenhänge ausnutzen, als Seitenkanal(SCA)-Angriffe. Seit ihrer Entdeckung haben sich SCA-Angriffe als äußerst effektiv erwiesen, um Schlüsselmaterial zu extrahieren und die Sicherheit vieler realer und weit verbreiteter Geräte zu brechen.

Aus diesem Grund wurde ein intensiver Fokus auf die Suche nach starken Gegenmaßnahmen gelegt. Einen prominenten und gut untersuchten Schutz gegen SCA-Angriffe bietet die Gegenmaßnahme *Masking*, die seit ihrer Einführung im Zusammenhang mit SCA vor zwei Jahrzehnten aufgrund ihrer soliden theoretischen Grundlage, die auf dem Konzept des *Secret Sharings* be-

---

ruht, den größten Teil der Aufmerksamkeit erhalten hat. Hier zielt eine zufällige Aufteilung der Informationen in den verarbeiteten Daten darauf ab, die erfolgreiche Wiederherstellung von geheimen Schlüsseln in den Bereich des praktisch Unmöglichen zu rücken.

Das Erreichen einer soliden Maskierung in Hardware ist eine langwierige und heikle Aufgabe, selbst für Experten, die auf viele Jahre Erfahrung im Design von SCA-resistenten Hardware Implementierungen zurückblicken können. Aufgrund der großen Vielfalt von physikalischen Effekten innerhalb digitaler Hardwareschaltungen können kleine Fehler im Design eine Implementierung in der Praxis völlig unsicher machen. Ausgehend von einem Prozess, der hauptsächlich die Maskierung auf der Grundlage von Heuristiken, und die experimentelle Bewertung der Sicherheit der resultierenden Hardware Implementierung durch gängige Evaluierungsmethoden umfasste, wurde ein Teil der Forschungsanstrengungen der Ableitung systematischerer Maskierungsansätze und der Bewertung von Hardwaredesigns in einem frühen Stadium des Designprozesses gewidmet. Die Erstellung formaler Angreifermodelle, die die Fähigkeiten eines Angreifers realistisch, aber abstrakt, modellieren und abdecken, war eine wichtige Voraussetzung für die Ableitung sowohl einer systematischen Maskierung, die sogar vollständig automatisiert werden kann, als auch einer Sicherheitsüberprüfung auf der Grundlage der Netzliste der Schaltung. Aufgrund seiner hohen Abstraktion ist das robuste  $d$ -Probing-Modell das am meisten untersuchte und ein gut geeignetes SCA-Angreifermodell im Kontext von Hardware.

Während wir generell darauf abzielen, Fortschritte im Hinblick auf die Gewährleistung eines soliden Schutzes gegen SCA-Angriffe zu erzielen, liefert diese Arbeit insbesondere Beiträge in drei Kategorien. Wir unterstreichen die anhaltende Relevanz von SCA-Angriffen - selbst für komplexe, modularisierte Systeme - indem wir zeigen, wie die Kombination von Software-Exploits und SCA eine starke Bedrohung für die Privatsphäre der Benutzer in weit verbreiteten Endverbrauchergeräten darstellen kann. Darüber hinaus stellen wir neue Techniken vor, um eine formale Verifikation zu realisieren, d.h. den Nachweis der Sicherheit und Verschaltbarkeit maskierter Hardware Designs in Bezug auf das formale robuste  $d$ -Probing Angreifer-Modell. Zu guter Letzt stellen wir mehrere neue Methoden zur systematischen und automatisierten Maskierung vor. Damit geben wir Ingenieuren einen Werkzeugkasten an die Hand, mit dem sie automatisch jede beliebige digitale Schaltung in eine Schaltung umwandeln können, die gründlich gegen SCA-Angriffe geschützt ist und gleichzeitig eine hohe Flexibilität bietet, um einen geeigneten Kompromiss zwischen Sicherheitsniveau, Latenzzeit und Platzbedarf des resultierenden Designs zu finden.

### **Schlagworte.**

Kryptographie, Hardwaresicherheit, Seitenkanalangriffe, formale SCA-Angreifermodelle, Maskierung, sichere Verschaltbarkeit, formale Verifikation, Automatisierung, iPhone Sicherheit

# Acknowledgements

This thesis is the result of around four years of research at the Chair for Embedded Security (EMSEC) headed by Prof. Dr.-Ing. Christof Paar, the Chair for Security Engineering (SECENG) headed by Prof. Dr.-Ing. Tim Güneysu, and the Implementation Security Group (IMPSEC) at the Ruhr-Universität Bochum under supervision of Prof. Dr. Amir Moradi.

I like to thank my family and friends. First and foremost, I want to thank my wife, Hannah. Without your emotional support, I could not have done it for sure. I met many intelligent people during this adventure, but you remain the most brilliant person I know. You mean the world to me. I love you. I also want to thank my parents, Sigrid and Jürgen. I would not be the person I am today if it was not for your unconditional support. Your love laid the groundwork for everything I have achieved. Thank you. I further like to thank my brother, Lukas, on whom I can rely without reservation and who likes to remind me that engineering is not real science. Furthermore, I want to thank my parents-in-law, Karin and Hassan, who treated me like family from the beginning and always offered refuge in their lovely home.

Next, I sincerely thank my advisor Amir Moradi for accepting me as a doctoral student and for his brilliant support. Your unwavering research interest and ability to push through boundaries are genuinely exceptional. Our many brainstorming sessions and deep discussions were the key enabler for this thesis. I appreciate that you always had an open ear, gave me a lot of freedom and trusted me concerning research and industry projects.

Moreover, I like to thank François-Xavier Standaert and Sebastian Faust for accepting to be referees for this thesis.

I am also thankful to Christof Paar and Tim Güneysu, who allowed me to be an employee of EMSEC and SECENG during my P.h.D. studies. I like to thank Falk Schellenberg for invoking my interest in hardware security by supervising my bachelor thesis and later offering me a position as a student research assistant. In the same course, I thank Bastian Richter for agreeing to supervise my master thesis and for many movie and game nights outside of work.

I also thank Jennifer Knothe and Janine Hein for their lovely support. Everyone knows we would be lost without you caring for everything behind the scenes.

Now on to the IMPSEC gang: I thank Thorben Moos for always being a fun office mate and calming me down after receiving my first reviews, Felix Wegener for awakening my entrepreneurial spirit and entertaining discussions about business ideas, and Anita Aghaie for lots of joint laughter. I want to thank Aein Razei Shamirzadi for being a great office mate and companion during this adventure, and Nicolai Müller, whose ‘getting-things-done’ attitude is utterly impressive. A big ‘thank you’ also to Marvin Staib, who, despite endangering my life by introducing me to bouldering, is just a lovely person. And to Bijan Fadaeinia, who always manages to cheer me up. A special shoutout also goes to Jannik Zeitschner, my tiger; there is much more to him than meets the eye. And to Daniel Lammers, who makes me want to get my exercise and nutrition right just by being present.

I also like to thank the entire SECENG team for fun social events (In particular, Flolle for being an outstanding boat crew member), great travel memories (In particular, Georg, Jakob and Jan for a great time in California), off-topic discussions (Told you, Markus, at some point I’ll get a mountainbike) and in general for the fantastic time, we all had together.

## Acknowledgements

---

I also like to thank my co-authors - in alphabetical order: Jakob Feldtkeller, Tim Güneysu, Oleksiy Lisovets, Thorben Moos, Amir Moradi, Nicolai Müller, Bastian Richter and Pascal Sasdrich. Thank you for all the great collaborations.

I wish all of you nothing but the best for your future.

# Table of Contents

Imprint . . . . .	v
Abstract . . . . .	v
Kurzfassung . . . . .	viii
Acknowledgements . . . . .	xi
<b>I Preliminaries</b>	<b>1</b>
<hr/>	
<b>1 Introduction</b>	<b>3</b>
1.1 Motivation . . . . .	3
1.2 Structure of this Thesis . . . . .	7
1.3 Our Research Contribution . . . . .	7
1.3.1 Persistent Relevance of SCA and Novel Threat Models . . . . .	7
1.3.2 Formal Verification of Masked Circuits . . . . .	9
1.3.3 Novel and Securely Composable Hardware Modules for Masking . . . . .	12
1.3.4 Methodologies and Tooling for Automated Masking . . . . .	15
<b>2 Background</b>	<b>17</b>
2.1 Notation . . . . .	17
2.2 Cryptography . . . . .	18
2.2.1 Asymmetric Cryptography . . . . .	18
2.2.2 Symmetric Cryptography . . . . .	18
2.2.3 Block Ciphers . . . . .	20
2.2.4 Practical Block Cipher Constructions . . . . .	20
2.3 Side-Channel Analysis Attacks . . . . .	21
2.3.1 Physical Attacks . . . . .	21
2.3.2 SCA Threat Model and Security Goal . . . . .	21
2.3.3 Power Side Channel . . . . .	22
2.3.4 Experimental Leakage Assessment . . . . .	23
2.3.5 Correlation Power Analysis . . . . .	25
2.4 SCA Countermeasures . . . . .	26
2.4.1 Hiding Countermeasure . . . . .	27
2.4.2 Masking Countermeasure . . . . .	27
2.5 Formal SCA Adversary Models . . . . .	28
2.5.1 Circuit Model . . . . .	28
2.5.2 Noisy Leakage Model . . . . .	29
2.5.3 Random Probing Model . . . . .	30
2.5.4 d-Probing Model . . . . .	30

## Table of Contents

---

2.5.5	Bounded Moment Model . . . . .	32
2.5.6	Relations Between Formal Adversary Models . . . . .	33
2.6	Automated Hardware Masking and Composability Notions . . . . .	34
2.6.1	Handcrafted Masking and Experimental Evaluation . . . . .	35
2.6.2	Towards Sound and Systematic Hardware Masking . . . . .	35
2.6.3	Gadget-Based Masking . . . . .	37
2.6.4	Composability Notions . . . . .	37
2.6.5	Gadget Realizations . . . . .	41
2.6.6	Discussions on the Costs of Automated Masking . . . . .	43
2.7	Formally Verifying SCA Resistance and Secure Composability . . . . .	43
2.7.1	Pre-Silicion Verification Methodologies for Hardware . . . . .	43
2.7.2	Binary Decision Diagram . . . . .	44
<b>II Publications</b>		<b>47</b>
<hr/>		
<b>3</b>	<b>Persistent Threat of Side-Channel Analysis Attacks</b>	<b>49</b>
3.1	Let's Take it Offline: Boosting Brute-Force Attacks on iPhone's User Authentication through SCA . . . . .	49
<b>4</b>	<b>Formal Verification of Masked Hardware Circuits</b>	<b>75</b>
4.1	SILVER - Statistical Independence and Leakage Verification . . . . .	75
4.2	Transitional Leakage in Theory and Practice – Unveiling Security Flaws in Masked Circuits . . . . .	107
<b>5</b>	<b>Novel and Securely Composable Hardware Modules for Masking</b>	<b>133</b>
5.1	Generic Hardware Private Circuits – Towards Automated Generation of Composable Secure Gadgets . . . . .	133
5.2	Composable Gadgets with Reused Fresh Masks – First-Order Probing-Secure Hardware Circuits with only 6 Fresh Masks . . . . .	157
5.3	Low-Latency Hardware Private Circuits . . . . .	187
<b>6</b>	<b>Methodologies and Tooling for Automated Masking</b>	<b>203</b>
6.1	Automated Generation of Masked Hardware . . . . .	203
<b>III Conclusions</b>		<b>247</b>
<hr/>		
<b>7</b>	<b>Conclusions and Open Problems</b>	<b>249</b>
<b>IV Appendix</b>		<b>251</b>
<hr/>		
<b>Bibliography</b>		<b>253</b>

<b>List of Abbreviations</b>	<b>265</b>
<b>List of Figures</b>	<b>267</b>
<b>List of Tables</b>	<b>269</b>
<b>About the Author</b>	<b>271</b>
<b>Publications and Academic Activities</b>	<b>275</b>





**Part I**

**Preliminaries**



# Chapter 1

## Introduction

*The scope and contributions of this work are outlined below. We discuss the importance of providing robust protection against side-channel attacks. In this context, we further motivate the need to formally verify hardware components regarding their side-channel resistance and secure integration into larger circuits. We also highlight the drawbacks of handcrafted, heuristic protection mechanisms in the context of masking, and emphasize the need for systematic techniques to achieve practical side-channel protection.*

### Contents of this Chapter

---

<b>1.1</b>	<b>Motivation</b>	<b>3</b>
<b>1.2</b>	<b>Structure of this Thesis</b>	<b>7</b>
<b>1.3</b>	<b>Our Research Contribution</b>	<b>7</b>

---

## 1.1 Motivation

The surge in demand for microchips, driven by the Internet of Things (IoT), Artificial Intelligence (AI), and the automotive transformation to autonomous driving, underscores the trend toward ubiquitous connectivity in a data-driven world. Its current scale far outstrips supply, which is further exacerbated by the increasing complexity of systems, shaky supply chains and a persistent talent shortage [BdJD22, BKP<sup>+</sup>22].

Considering autonomous driving as a highly relevant example of large-scale communication networks based on real-time data collection and processing, secure vehicle-to-vehicle communication in compliance with fundamental security goals such as confidentiality, integrity, authenticity and availability is crucial to ensure user safety and privacy. For performance reasons, such complex systems, which involve a large number of communication nodes, use heterogeneous cryptographic primitives, typically combining asymmetric cryptography for the establishment of a secure communication channel and symmetric cryptography for the actual high-throughput communication over this channel.

While today's established ciphers are considered secure in a purely cryptanalytic sense, i.e., there are no known possible key recovery attacks in a black-box scenario where the adversary is limited to choosing arbitrary plaintexts and observing the corresponding ciphertexts without gaining any information about the computation itself, their real-world integration into larger

systems has proven to be highly inconsistent with these assumptions. This leaves the implementation and integration of cryptographic algorithms vulnerable to broad physical attack vectors in many scenarios, thus compromising security and potentially having far-reaching consequences with respect to a system's security and privacy guarantees.

Even though there has been a long and deep research effort to protect cryptographic implementations against these types of physical attacks, the countermeasures themselves and the process of thoroughly protecting implementations are still far from reaching their optimum in terms of efficiency and cost-effectiveness, ensuring a continued high relevance for achieving further improvements.

Consequently, several factors are crucial to consider in this context. A practical and continuous assessment of the relevance of physical attacks and their threats is important to cover the majority of attack vectors and scenarios. Furthermore, it is essential to reduce the cost of implementing solid countermeasures, and it is highly beneficial to optimize the development process, since resources in terms of money, time and expertise are tied up in realizing thorough protection against these types of attacks. This thesis aims to make progress on all of these factors.

**Silicon Root of Trust.** The integrity of a system and the guarantee that it will boot in a secure and trusted state is rooted in the hardware components involved and the secret key material stored on them. Therefore, it is crucial that the keys remain secret and unaltered, even in the presence of physical access, such as during shipping. A system's trust is therefore anchored in the lowest level of its components: The silicon of the hardware. While the majority of chip vendors offer some form of silicon root of trust, Google and its partners have launched the *OpenTitan* project [JRR<sup>+</sup>18] with the goal of developing an open source root of trust through design and implementation transparency, underscoring the strong need for a trustworthy hardware foundation. Since the security guarantees of a complex system ultimately fall back on the security of the hardware, it is crucial to build a silicon root of trust that is thoroughly protected against all kinds of physical attack vectors.

**Cryptographic Hardware Accelerators.** Driven by increasing functionality requirements and decreasing technology size, Integrated Circuits (ICs) and Printed Circuit Board (PCB) designs are becoming increasingly complex, making it difficult to perform thorough security evaluations. To ensure high throughput and confidential communication between devices, general-purpose computing units are often paired with dedicated symmetric encryption logic, either in a completely separate IC or on the same die in the form of a System on Chip (SoC). Examples of such accelerators are Intel's and AMD's AES-NI [HC12], but Apple's iPhones are also equipped with dedicated AES hardware realizations placed within their Application Processor (AP) and Secure Enclave Processor (SEP) [App22], which are crucial components in the root of trust and are involved in the boot, user authentication and data encryption processes.

**Cost of Hardware Development and Production.** Security integration and evaluation are important factors to consider when estimating the development and production costs of a new IC design. In general, the cost of manufacturing and developing ICs can be broadly divided into one-time costs and recurring costs per IC. Minimizing fixed, non-recurring costs is important for

---

low-volume and highly specialized ICs. According to this categorization, security-related non-recurring costs include labor costs for expensive security specialists and security evaluation. At the same time, the increased area required to integrate protection mechanisms drives recurring production costs. Finding new ways to optimize both is a highly relevant goal of ongoing academic research and the semiconductor industry. It is therefore crucial not only to thoroughly protect the hardware implementation against physical attacks, but also to establish a scientific basis for implementing countermeasures in an economically viable way, thus lowering the barrier for integrating important protection mechanisms. Such a lower burden will lead to a wider application of robust protection, even for devices of smaller product lines.

**Side-Channel Analysis Attacks.** At a high level, SCA refers to attacks in which the adversary exploits unintended dependencies between sensitive data stored on or processed by a cryptographic device and the physical characteristics of the device. The term *cryptographic device* simply describes a device that performs some form of cryptographic operation, like an encryption, or stores cryptographic key material [MOP07]. Cryptographic devices include general-purpose microcontroller, reconfigurable hardware (Field Programmable Gate Arrays (FPGAs)) and Application Specific Integrated Circuits (ASICs) – possibly in combination. While there are many more, common examples of exploitable features include timing [Koc96], power consumption [KJJ99] and electromagnetic radiation [GMO01]. Passively observing and analyzing these physical properties often allows an adversary to completely extract secret key material from smart cards [MOP07], smart phones [BFMT16, VMC19, LKMM21] and other critical hardware systems [EKM<sup>+</sup>08, KKMP09, OP11, ORP13, SRH16]. Although the threat model initially assumed direct physical access to the device, more and more research has been published that relaxes this requirement by introducing fully remote attack vectors [SGMT18, ZS18, SGMT21], further expanding the scope of dangerous attack scenarios and highlighting the need for adequate protection mechanisms. In this thesis, we focus on power-related SCA on symmetric algorithms realized as a digital hardware circuit.

**Masking as a SCA Countermeasure.** In general, mitigation strategies for SCA aim to weaken the dependency between confidential data and the leaking physical characteristics (the side channel) to such an extent that obtaining useful information becomes impractical for any realistic adversary. Besides *hiding* [MOP07], which relaxes this dependency more directly through technical means, e.g., by increasing electrical noise or equalizing power consumption for different secret states or operations, *masking* is a prominent and well-studied SCA countermeasure on an algorithmic level [ISW03, Tri03, NRS11, RBN<sup>+</sup>15, GMK17, GM18, SM21]. Rooted in the concept of secret sharing [Sha79], sensitive data is split into independent, random shares, and the cryptographic operation is then performed on the shares instead of the secret itself. Intuitively, this splitting of information makes recovery much more difficult (and, if properly implemented, even prohibits practical SCA altogether), but it is non-trivial to implement and introduces high overhead into the design in terms of latency and area footprint.

**Formal Adversary Models.** To formally argue about the resilience of a hardware design against SCA, various adversary models have been introduced over time, aiming to realistically model a real-world SCA adversary while working at different levels of abstraction [ISW03, FGP<sup>+</sup>18, DDF19, BDF<sup>+</sup>16, DFS15]. The main advantage of introducing formal

adversarial models is twofold. First, given sufficient abstraction, it allows for the systematic formulation of new masking algorithms and methods in provable compliance with security in the adversary models. Second, it allows evaluating the security of a design with respect to the formal adversary model. This security assessment can be made very early in the design cycle, on the synthesized netlist of a masked implementation. Assuming that the model accurately covers a real adversarial scenario, practical SCA resilience can be guaranteed or verified prior to production, eliminating the need for multiple iterations after experimental security verification fails, and as a consequence drastically reducing cost and time to market.

**Securely Composable Gadgets and Automated Masking.** Masking large circuits that realize complex, nonlinear functions such as ciphers has proven to be extremely difficult. Even for experts with years of experience in SCA and masking countermeasures, the process of masking such designs is highly error-prone. Since minor errors can render a design completely insecure in practice [MMSS19], a major branch of research with respect to masking focuses on establishing more systematic and automatable ways to achieve SCA resilience [ISW03, BBD<sup>+</sup>16, CS20]. Here, with a solid theoretical foundation based on formal adversary models, masked hardware modules have been developed that guarantee SCA resilience even when interconnected to form a larger circuit. These building blocks – commonly referred to as hardware *gadgets* in the literature – are then used to mask any given circuit in a divide-and-conquer fashion. The underlying process can even be completely automated. Many of these building blocks realize masked versions of atomic functions like an AND gate. Still, they can be instantiated for arbitrary security levels, eventually enabling the systematic masking of arbitrary digital circuits at any desired security level.

**Formal Verification.** The traditional approach to verifying an implementation’s resistance to SCA relies heavily on experimental security verification after IC fabrication. The goal of this experimental verification is to identify any dependencies between a given physical characteristic (such as electromagnetic radiation or power consumption) and a secret currently stored on or processed by the device. This is done by statistical analysis of actual experimental measurements. Since this type of practical verification requires going through all the steps of the hardware design process, including layout and fabrication, a design that turns out to be insecure in practice must be adjusted before fabrication is repeated. Obviously, this iterative process is very costly and time consuming. Accurate formal adversary models make it possible to evaluate the level of protection against SCA very early in the design process. Typically, it can be performed at the netlist level after the Register Transfer Level (RTL) description has been synthesized with respect to a specified gate library, eliminating the need for a try-and-error approach and drastically reducing the resources required in terms of time and cost. Assuming that the adversary model accurately captures all realistic SCA attack scenarios, software tools for formally verifying the security of an implementation through its netlist are highly beneficial for achieving SCA resilience in an effective and efficient manner [BGI<sup>+</sup>18, BBC<sup>+</sup>19, KSM20, MM22, BMRT22].

**Cost Reduction and Use-Case Specific Requirements.** While achieving SCA resilience through masking is an important goal in itself, making it practical, i.e., achieving it at a reasonable cost and guaranteeing sufficient throughput and speed, is ultimately the more relevant

objective. Compared to handcrafted masking based on heuristics and experience, fully automated masking typically introduces significant overhead into the design in terms of randomness requirements, additional latency and the area footprint of the protected circuit. The area footprint (and randomness requirements since some sort of randomness source with sufficient throughput must be provided on-chip) translates directly into the cost of producing cryptographic ICs. At the same time, low latency is important for specific use cases such as fast memory encryption. The creation of an improved and diverse library of hardware gadgets is highly important to enable automated masking with reduced production costs and to tailor the protected design to a specific use case.

## 1.2 Structure of this Thesis

This thesis is divided into three main parts. The first chapter in Part I contains the introduction to our work, giving a general motivation, followed by this section, and finally summarizing the main research contributions in Section 1.3. The following chapter of this part, Chapter 2, provides all the scientific concepts used throughout this thesis, including elaborations on symmetric cryptography, side-channel analysis and countermeasures, formal side-channel adversary models, automated masking and composability notions as well as formal verification of side-channel resistance. Part II contains all the scientific publications that are part of this thesis. Each publication is preceded by a brief summary of its content, the publication details and a summary of the contributions made by the author of this thesis. This chapter is divided into four topical sections that form the basis of this thesis. Chapter 3 deals with the persistent threat of side-channel analysis, before publications dealing with the formal verification of masked hardware circuits are presented in Chapter 4. This is followed by our research on new composable hardware modules for masking in Chapter 5 and our publication on novel methods and tools for automated masking in Chapter 6. Part III summarizes this work and discusses open issues that may be addressed in future work.

## 1.3 Our Research Contribution

Generally, all research contributions made by the scientific publications forming this thesis aim to effectively and efficiently protect hardware against power-related SCA attacks. This objective is founded on four pillars. First, we elaborate on new SCA attack scenarios in the context of widespread end-consumer targets. Second, we introduce new methodologies and tools for verifying a hardware design's resilience against SCA solely based on the netlist representation of the circuit. Subsequently, we present a set of novel masked hardware modules that can be freely combined without compromising security guarantees, before we present new techniques for enabling and improving the automated generation of masked circuits.

### 1.3.1 Persistent Relevance of SCA and Novel Threat Models

A continuous evaluation of possible new threat scenarios with respect to physical attacks in modern and complex hardware architectures is crucial to ensure high security of widely used devices. Therefore, part of the focus of this thesis is to elaborate on the relevance of side-channel

attacks on complex and modularized consumer devices that are based on various hardware and software components.

**Advances in SCA Attack Scenarios on Widespread Consumer Devices [LKMM21].** Published work on practical power side-channel attacks on consumer devices has become rare due to the immense increase in complexity of modern ICs and SoCs, accompanied by the increased difficulty of performing successful SCAs attacks. Smartphones are good examples of such complex architectures. We show that when it comes to the security of complex systems, hardware and software cannot be considered independently, and that there are manifold attack vectors that combine both to form a powerful lever for privacy exploitation.

To ensure that a complex system boots into an intended state, the boot process is divided into several steps that together form a chain of trust. Each step guarantees the integrity of the next. This chain typically starts with a minimal piece of software embedded in the chip that cannot be updated by firmware updates. This immutable piece of software is used to validate the integrity of any software that is subsequently loaded. Together with the hard-fused key material used for validation, this piece of software forms the root of trust. Changing this chain or obtaining key material from the hardware can have immense consequences for the security of a system, as it allows the boot process to be altered and modified software to be loaded. Therefore, in complex environments, security analysis of hardware and software should never be considered separately.

In our work, we use a public BootROM exploit on an iPhone 4 to run our own software on the device, with the goal of gaining oracle access to the iPhone’s AES core. By oracle access, we mean that we can trigger an unlimited amount of encryptions and decryptions on chosen input data. This oracle access allows us to perform SCA attacks on the iPhone 4’s hardware-fused key material. We perform an SCA based on the Electro-Magnetic (EM) emanation of the AP as well as an attack based on the imminent power consumption. In addition to the Group Identifier (GID), a key that is identical across devices of the same model and is involved in guaranteeing the integrity of the operating system, we successfully recover the so-called Unique Identifier (UID) key, which is device-specific and plays a crucial role in user authentication. More specifically, it cryptographically binds the user authentication to be performed on the device. This process is intentionally designed to take some time (about 80ms [App12]) to create a hardware barrier to brute forcing the user’s passcode. Once the key is recovered, the passcode search can be performed offline and is arbitrarily scalable with respect to the underlying resources. Without any advanced optimizations, we show that using multiple Graphical Processing Unit (GPU)s dramatically accelerates the brute force search compared to on-device computation. For example, we reduce the worst-case runtime for a common six-digit number from 22 hours to only 26 seconds using 8 GeForce RTX2080 TIs [NVI18].

Thus, with this work, we highlight the importance of continuously evaluating consumer devices for possible implementation attacks and implementing appropriate countermeasures that protect against all types of attack scenarios in the context of complex systems involving various hardware and software components. We do this by presenting the following main contributions:

- (i) *Successful recovery of the GID and UID key from an iPhone 4 by performing a Correlation Power Analysis (CPA) attack. A picture showing the setup for the CPA, including the EM probe used, can be seen in Figure 1.1.*



- (ii) *Significant gain over an on-device brute-force passcode search, boosted by several GPUs, since the extraction of the UID allows it to be performed offline. The results are given in Table 1.1.*

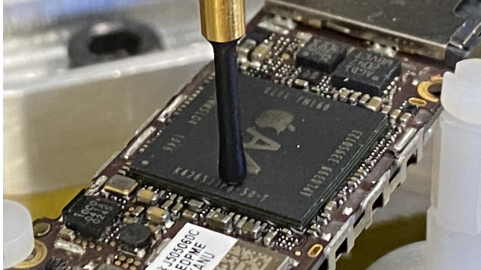


Figure 1.1: CPA setup: EM probe placed on the AP of the iPhone 4 [LKMM21].

digits	iPhone	1xGPU	8×GPU
4	13 minutes	2 seconds	< 1 second
6	22 hours	3 minutes	26 seconds
7	9 days	35 minutes	4 minutes
8	92 days	5 hours	43 minutes
9	925 days	58 hours	7 hours
10	25 years	24 days	3 days
11	253 years	243 days	30 days
12	2536 years	2439 days	304 days

Table 1.1: Runtime comparison of passcode search: on-device vs. GPU cluster. GPU = RTX 2080 TI [LKMM21].

### 1.3.2 Formal Verification of Masked Circuits

Traditionally, evaluating the resilience of cryptographic hardware implementations against SCA attacks has been, and often still is, purely experimental. However, a purely experimental approach has the disadvantage that production must be fully completed before a core can be evaluated. If the evaluation indicates an inadequate level of security, the design must be adjusted accordingly. This iterative methodology can be extremely costly and time-consuming. In addition, it is immensely difficult to identify the source of the vulnerability by analyzing the results of the experimental evaluation alone.

Using formal adversary models, we can evaluate the SCA resilience of a cryptographic hardware core very early in the design process, at the netlist level of a circuit. In contrast to an experimental setup, formal adversary models further abstract from a specific technology, platform and environment, hence allowing for a worst-case security evaluation. While experimental security guarantees are always in the context of the used evaluation setup and environment, formal adversary models enable a more general security statement.

In this thesis, we therefore introduce novel verification techniques to guarantee a thorough SCA resilience early in the design process of a new cryptographic implementation.

**A Complete and Accurate Tool for Verifying Security and Composability in the Robust  $d$ -Probing Adversary Model [KSM20].** We have develop a novel methodology for analyzing security and composability in the formal  $d$ -probing model [ISW03] and its robust variant [FGP<sup>+</sup>18]. Our technique allows to perform thorough SCA resilience checks at the netlist level of an implementation, thus detecting security flaws early in the circuit design process. To achieve this, we translate all security and composability notions into a unified form based on evaluating the statistical independence between observations and (unshared) circuit inputs. We then use so-called Binary Decision Diagrams (BDDs) [Jr.78] which are well-established data structures for circuit analysis performed in the context of Electronic Design Automation (EDA). These data structures allow us to check for such statistical independence and thus evaluate security with respect to the formal security and composability notions considered.

We have implemented our results in the form of a tool called SILVER. At the time of publication, SILVER was the first *complete* and *accurate* tool for security verification in the context of the formal glitch-extended robust  $d$ -probing model. It is *complete* in the sense that it covers all common formal notions of security and composability. It is *accurate* in the sense that it does not produce false positive or false negative results. Of course, if a design is claimed to be secure, it is crucial that it really is secure, i.e., that it conforms to the security definition of the  $d$ -probing model. On the other hand, other existing formal verification tools, such as maskVerif [BBC<sup>+</sup>19], allow false negatives, i.e., designs that are claimed to be insecure by the tool are actually secure in the  $d$ -probing model. This is because the security check is performed in an overly conservative manner to achieve a reduction in verification complexity. The downside of allowing such false negatives is that it can introduce unnecessary overhead into a circuit, because in practice an engineer will tweak the circuit until the tool indicates its security. With SILVER, we get an accurate security statement at the cost of reduced efficiency. For this reason, SILVER is most useful for evaluating smaller hardware designs in the context of composability notions.

In addition to indicating the security and secure composability of a hardware circuit, SILVER provides additional information to the user. If the circuit is insecure or non-compliant with one or more composability notions, SILVER provides the cause of the problem in the form of leaking probes. This is a very useful piece of information, as it makes it much easier to locate the source of the problem. In practical leakage assessments, it is expected that the noisy sum of all wires will be observed during a single clock cycle [CJRR99]. Since many operations are performed in parallel during a single clock cycle in hardware, the exact leakage source cannot be detected by a classical Test Vector Leakage Assessment (TVLA).

With this work, we have further improved the formal verification of hardware circuits. In particular, the accuracy and the support of all common composability notions have proven to be very useful in practice and have paved the way for much follow-up work on automation and efficient designs – by us and within the research community. With our work we achieve the following main contributions:

- (i) *Unifying the methodology for checking security and composability notions in the glitch-extended robust  $d$ -probing model using statistical independence.*
- (ii) *Introducing SILVER, a software tool that, based on a circuit's netlist, can evaluate security in the glitch-extended  $d$ -probing model along with all common composability notions (Non-Interference (NI) [BBD<sup>+</sup>15], Strong Non-Interference (SNI) [BBD<sup>+</sup>16] and Probe Isolating Non-Interference (PINI) [CS20]). It is also capable of checking the output uniformity [Bil15] of a circuit.*
- (iii) *SILVER provides precise information about the origin of the leakage.*
- (iv) *As SILVER works on the synthesized netlist of a hardware design, it is perfectly suited to be integrated into established EDA design flows and tool chains.*

### **Covering Data Transitions on Registers in the Context of Formal Verification [MKSM22].**

Data transitions at registers are known to be a major source of information leakage in hardware designs. Therefore, they are abstractly covered in a worst-case manner in the context of the (robust)  $d$ -probing model [FGP<sup>+</sup>18]. This abstraction allows for a simple security check under

Scheme	Latency	Randomness	Function	Reference
HPC1	2	$d(d+1)/2 + r[d]^\dagger$	AND	[CGLS21]
HPC2	2	$d(d+1)/2$	AND	[CGLS21]
<b>GHPC*</b>	2	$m$	$f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$	[KSM22]
<b>GHPC<sub>LL</sub>*</b>	1	$2^n \cdot m$	$f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$	[KSM22]
<b>HPC3</b>	1	$d(d+1)$	AND	[KM22b]

$^\dagger r[d] = [1, 2, 4, 5, 7, 9, 11, 12, 15, 17]$  for  $d \leq 10$

\* restricted to  $d = 1$

Table 1.2: Comparison of existing PINI-composable and glitch-robust Hardware Private Circuits (HPCs). Gadgets in **bold** were published in the course of this thesis.  $d$  denotes the security order.

transitions by simply adding an additional preprocessing step to the probing sets placed on a sequential circuit.

While formal hardware verification tools had already covered glitches as a common physical effect in hardware that facilitates leakage, consideration of transitions at registers and the combination of transitions and glitches remained very rudimentary. Since our initial methodology for SILVER also lacks any consideration of transitions, we extend our initial work and include support for transitions in the original version of the tool.

More precisely, we extend SILVER to support sequential, iterative circuits, where the same physical module can be executed multiple times with respect to the same input, and where a sequence of different inputs must be considered accurately. We also adapt our probe extension methodology to adequately cover the occurrence of transitions and the combination of glitches and transitions. This extension allows us to analyze various designs found in the literature and identify leakage caused solely by transitions at registers. We are able to confirm our tool-based findings by performing practical elaborations on top.

With this work, we highlight the importance of considering transitions when formally verifying hardware designs. We show that, while transitions lead to insecurity when not considered appropriately, adding a lot of additional overhead into the design just to handle transitions can be unnecessary in many cases. With our new version of SILVER, we are able to circumvent both cases, further paving the way for efficient and effective masking countermeasures.

Our main contributions are summed up in the following:

- (i) *We derive a new technique for handling transitions and the combined occurrence of glitches and transitions for iterative circuits that can be fed by a sequence of different inputs.*
- (ii) *We extend SILVER to support our new methodology and study various masked circuits and schemes from the literature for their (in)sufficiency in handling transitions appropriately.*

### 1.3.3 Novel and Securely Composable Hardware Modules for Masking

Securely masking large and complex circuits is a delicate and error-prone task, especially when higher security orders are considered. Therefore, securely composable hardware modules – commonly referred to as *gadgets* – can provide a crucial building block for creating larger hardware components that are thoroughly protected against SCA attacks in a divide-and-conquer fashion. In the course of this thesis, we have developed several new gadget constructions, all of which offer different trade-offs with respect to the overhead metrics of a masked circuit. Consequently, the contributions made in this part of the thesis aim to give a researcher or engineer more flexibility to tailor a masked circuit to a specific use case. For example, reducing latency as much as possible may be a requirement when implementing memory encryption. Another requirement might be to make the masking as inexpensive and therefore as small as possible, while still achieving a very high level of security. All our designs aim to provide more freedom in the systematic design of masked circuits.

**Extending the Functionality of Trivially Composable Hardware Modules [KSM22].** Until the publication of our work, freely composable gadgets were limited to implementing very basic functionality. They typically represented masked variants of atomic logic gates, such as a simple two-input AND or OR.

Being restricted to such gadgets with limited functionality has the significant drawback that each gadget introduces individual overhead – such as randomness requirements, area footprint and latency – into a design. Since a large number of gadgets is required to realize complex circuits such as entire encryption cores, this naturally results in a significant final overhead for the protected implementations.

In our work, we focus on extending the functionality that composable gadgets can provide while not drastically increasing the randomness and latency overhead over existing atomic gadgets. With our novel construction called Generic Hardware Private Circuit (GHPC), we introduce a methodology to realize gadgets that are freely composable under the notion of PINI [CS20] in the glitch-extended robust probing model with arbitrary functionality. Such gadgets can be easily derived based on the functional description, i.e., the vectorial Boolean function, it should realize. While restricting our considerations to the first security order, we introduce two variants of our technique.

In its standard variant, GHPC allows to realize any functionality with only two register stages and one fresh random bit per coordinate function. Our low-latency variant – denoted GHPC<sub>LL</sub> – allows to build a gadget for any functionality with  $n$  unshared inputs, using only a single register stage, but consuming  $2^n$  random bits per coordinate function. A comparison with other PINI-composable hardware gadgets can be seen in Table 1.2. It should be noted that the area complexity grows exponentially with the number of inputs to the circuit. For this reason, it is advantageous to avoid building very large functions as GHPCs, but to adjust the tradeoffs between latency, randomness and area requirements by adjusting the complexity of the gadgets in terms of their functional scope. For example, we can realize a complete S-box as a single GHPC with a rather large area footprint but low latency and low randomness requirements, or we can reduce the area by further dividing the S-box into functional blocks and then building GHPCs for those blocks.

We also like to emphasize that our introduced constructions are freely composable with any other existing PINI-composable gadget, thus further extending their range of application.

As a result, our novel GHPC constructions hand an engineer more flexibility for masking a circuit based on composable gadgets, and allow the creation of a custom gadget library where gadgets can be selected based on the use case and what trade-off between fresh randomness, latency and area footprint is required.

Our work provides the following key advantages over the state of the art:

- (i) *We introduce GHPC – our standard variant: A generalized construction that allows building first-order, PINI-composable hardware gadgets for arbitrary vectorial Boolean functions. This variant requires only a single fresh random bit per coordinate function and has a latency of two clock cycles.*
- (ii) *We introduce GHPC<sub>LL</sub> – our low-latency variant: A generalized technique for building first-order, PINI-composable hardware gadgets for arbitrary vectorial Boolean functions. This variant requires  $2^n$  fresh random bits per coordinate function, where  $n$  is the number of unshared inputs, while requiring only a single cycle latency.*

**Freely Composable Gadgets with Randomness Reusage [KM22a].** The randomness requirements of masked circuits ultimately translate into area overhead, since a dedicated randomness source must be placed on-chip. This source must be able to provide sufficient throughput for the required randomness supply. To ensure a secure design, each random bit supplied to a composed circuit must be renewed every clock cycle in the context of common and widely used gadget designs and composability notions. Because a large number of gadgets is typically required to implement full encryption – each with individual randomness requirements – gadget-based masking typically results in designs that require a high-throughput randomness source.

While a generic cost function for generating randomness has never been derived, meaning that there is no lower bound on the area required to implement a randomness source capable of providing a given throughput, it is an interesting topic to reduce the randomness requirements of masked circuits and evaluate the resulting overall area gain in terms of a common implementation of a randomness source.

One way to reduce the overall randomness requirements of a circuit is to construct gadgets that allow the reuse of random bits across gadget boundaries.

In our work, we introduce COMAR, a methodology that leads to a set of novel gadget constructions that implement all common atomic logic gates in a masked fashion, and allow us to construct any circuit that is protected for the first security order in the glitch-extended robust probing model. This allows us to reduce the number of required fresh bits to a total of six, completely independent of the complexity and size of a circuit. This means that we can build any protected circuit using a total of only six fresh random bits, not including the randomness needed to share the input to the circuit.

While paying the price of increased latency, we show that if the randomness supply is implemented as a Linear Feedback Shift Register (LFSR), which is common practice in comparable literature, we can achieve significant reductions in the area consumption of the overall design, i.e., when considering the encryption core together with the LFSRs.

In addition, we show that the input width of our COMAR gadgets can be arbitrarily increased without significantly increasing the overall randomness requirements. This is simply due to the

fact that the total requirements are only determined by the maximum requirements of a single gadget, since all random bits can be reused across gadgets. Widening the input width does not increase the latency of a gadget, but it generally allows the design to be fine-tuned for a better latency/area tradeoff.

With COMAR, we offer another tool for systematic masking that effectively decouples the randomness requirement of a first-order masked circuit from its size and complexity, allowing arbitrary circuits to be masked using a total of only six random bits. This is another step in the direction of giving a designer more flexibility to tailor a circuit to a particular use case.

Our work makes the following main contributions to freely composable gadgets with randomness reuse:

- (i) *A freely first-order composable AND gadget that can be instantiated for arbitrary input widths. Gadgets for other nonlinear logic gates can be trivially derived from this AND gadget.*
- (ii) *A freely first-order composable XOR gadget, also instantiable for arbitrary input widths.*
- (iii) *Random bits are fully reusable across gadgets. If the circuit instantiates nonlinear COMAR gadgets with at most  $n$  inputs and linear COMAR gadgets with at most  $m$  inputs, the number of fresh mask bits required for the entire circuit is  $\max(n, m) + \max(2n, 2m)$ . If we only use two-input gadgets, this results in an overall requirement of only six fresh random bits.*

**Low-latency Hardware Private Circuits [KM22b].** Latency – in terms of the number of clock cycles required for an input to produce the desired output – is an important factor in synchronous circuit design. Low latency is desirable in many applications, such as fast memory encryption. Glitches in the robust  $d$ -probing model are one of the main drivers of latency, as registers must be carefully placed to avoid unwanted information leakage due to signal recombinations caused by such physical effects.

Since composability notions limit leakage for each gadget individually, each gadget instantiation introduces latency into the design. In the context of PINI [CS20], the overall latency is determined by the path within the circuit that contains the most nonlinear atomic gates, since linear operations can be performed in a trivial, share-wise manner and do not contribute additional latency to the design.

Intuitively, the length of such a path is determined by the algebraic degree of the Boolean function the circuit is realizing, and is quite large for cryptographic primitives, for example in the context of AES, which uses an 8-bit S-box. While [CGLS21] shows how global optimizations in the context of HPC2 gadgets can actually reduce the overall latency for small 4-bit S-boxes by taking into account the asymmetry in the input-output latency of the gadgets, no PINI-composable gadget had a single clock cycle latency for all inputs.

In our work, we introduce HPC3, a PINI-composable gadget that realizes a masked variant of an AND gate and is instantiable for arbitrary security orders. Compared to the state of the art, HPC3 has a latency of only a single clock cycle, making it highly advantageous in low-latency use cases. The decrease in latency we achieve comes at the cost of doubling the randomness requirements compared to HPC2. An overview of PINI composable gadgets is presented in Table 1.2.

We also present a variant of HPC3 that is composable under the Output Probe-Isolating Non-Interference (O-PINI) notion introduced in [CS21] with the aim of guaranteeing trivial composition under the combined occurrence of glitches *and* transitions. As evaluated as part of this thesis [MKSM22], its practical use is mainly limited to specific edge use cases.

As a result, our work offers new constructions for the systematic creation of masked circuits, which is advantageous in use cases where latency is a critical factor. The main advances of our work are as follows:

- (i) *We present HPC3: A PINI-composable hardware gadget that implements a masked variant of an AND gate and can be initialized at arbitrary security orders. It has a latency of only a single clock cycle, while doubling the randomness requirements compared to the state of the art, i.e., HPC2 gadgets.*
- (ii) *This allows us to achieve a significant latency reduction for gadget-based masking compared to comparable work.*

### 1.3.4 Methodologies and Tooling for Automated Masking

Securely masking large and complex circuits is a difficult task, especially when considering higher security orders. It requires extensive experience and time. Achieving solid protection through handcrafted masking based solely on heuristics is extremely difficult for higher security orders, which is underscored by the lack of published work presenting such heuristically derived designs that are secure for security orders greater than three.

Therefore, a systematic and automatable masking approach based on a divide-and-conquer methodology that allows to mask arbitrary circuits at any desired security order has been part of the focus within the research community. Such a methodology is based on composable hardware modules, of which we present several novel variants in this thesis.

The simple underlying idea is based on the replacement of unprotected gates by their masked and securely composable counterparts: The corresponding hardware gadgets. Since these counterparts fulfill well-defined theoretical properties that guarantee security when composed together, the derived circuit will be provably SCA resistant with respect to the (robust)  $d$ -probing adversary model.

**Automated Masking of Unprotected Circuits [KMMS22].** In our work, we present a tool for automated hardware masking that effectively relies on a simple divide-and-conquer approach.

Our tool, called AGEMA, is based on composable gadgets. In fact, AGEMA works on a custom library of PINI-composable gadgets that can be easily extended by any novel constructions, as long as they fulfill the notion of PINI [CS20]. Basically, AGEMA takes the netlist of an unprotected circuit as input and outputs a circuit that is secure in the formal glitch-extended  $d$ -probing model [FGP<sup>+</sup>18]. It is possible to configure various parameters that define not only how the unprotected circuit is preprocessed before being masked, but also the security order. In general, it is possible to mask a circuit for any desired security order, while arbitrary security orders are not possible for all preprocessing methods. Since gadgets introduce individual latencies into the design, AGEMA users can choose between two different synchronization techniques. If higher throughput is desired, the design is pipelined, while clock gating can be used to achieve a more compact circuit design.

An extensible library of gadgets, together with our various preprocessing methods, enables the rapid generation of protected hardware, while providing many options for customizing the implementation to best suit the needs of a specific use case. This allows engineers to find favorable tradeoffs between randomness requirements, area footprint and latency. Integration of future, novel PINI-composable gadgets is straightforward, as they can be easily integrated into the custom gadget library.

At the time of publication, AGEMA was, to the best of our knowledge, the first fully automated tool for turning any entirely unprotected hardware design into a design thoroughly protected against SCA attacks, offering provable security guarantees in the context of the (robust)  $d$ -probing model. We should highlight that the process only needs a few seconds.

As a consequence, with our work, we present the following main advances:

- (i) *We introduce AGEMA: A tool for fully automated masking of unprotected circuits. The result is guaranteed to be secure in the glitch-extended robust  $d$ -probing model. AGEMA is able to achieve security for arbitrary security orders.*
- (ii) *Introduction and thorough comparison of different preprocessing techniques for the unprotected circuit and elaboration of their influence on the resulting, masked circuit.*
- (iii) *AGEMA is well suited for direct integration into existing EDA tool chains.*



# Chapter 2

## Background

*In this chapter, we provide the scientific foundation necessary for our work. We give an introduction to physical attacks, in particular SCA attacks. In this context, we will give a detailed description of existing abstract adversary models and explain the masking countermeasure, before describing the concept of composability notions and how they can be used for systematic and automated masking. Finally, we will show how the SCA resilience of a hardware design can be formally verified in the context of the corresponding adversary model.*

### Contents of this Chapter

<b>2.1</b>	<b>Notation</b>	<b>17</b>
<b>2.2</b>	<b>Cryptography</b>	<b>18</b>
<b>2.3</b>	<b>Side-Channel Analysis Attacks</b>	<b>21</b>
<b>2.4</b>	<b>SCA Countermeasures</b>	<b>26</b>
<b>2.5</b>	<b>Formal SCA Adversary Models</b>	<b>28</b>
<b>2.6</b>	<b>Automated Hardware Masking and Composability Notions</b>	<b>34</b>
<b>2.7</b>	<b>Formally Verifying SCA Resistance and Secure Composability</b>	<b>43</b>

### 2.1 Notation<sup>1</sup>

Capital letters usually denote random variables. In the context of masking, we indicate share indices by subscripts, while input indices are identified by superscripts, e.g.,  $X_i^j$ , denotes the random variable describing the  $i$ -th share of the  $j$ -th input. Calligraphic font is used to denote sets, while vectors are written in bold. We usually denote the number of shares as  $s$ . We denote the vector of all valid Boolean sharings of a specific value  $x \in \mathbb{F}_2$  by  $Sh(x) = \{\mathbf{x} = (x_0, x_1, \dots, x_{s-1}) \mid \bigoplus_{i=0}^{s-1} x_i = x\}$ . We denote all shares from a set  $\mathcal{S}$  corresponding to input  $j$  by  $\mathcal{S}^j$ . While we use  $Pr[\cdot]$  for denoting the probability, we otherwise use common notation for statistical concepts like dependencies. We use  $E(\cdot)$  for the expectation operator. Assignments to variables – that are possibly the result of a randomized processes – are indicated by ‘ $\leftarrow$ ’. If we want to highlight a deterministic assignment, we use ‘ $:=$ ’. We further denote drawing  $X$  independently and uniformly from  $\mathcal{R}$  by  $X \stackrel{\$}{\leftarrow} \mathcal{R}$ .

<sup>1</sup>May differ from the notation used in Part II. The notation of the included publications is always declared within the publication.

## 2.2 Cryptography

While the concept of secret messages goes back a long way, and rudimentary encryption devices existed as early as ancient Greece, a more formal and broader consideration emerged in the early 20th century and was accelerated by the introduction of computers and the digitization [Dav97]. While cryptography was originally concerned with keeping messages between two parties secret in an information-theoretic sense, modern-day cryptography is a vibrant area of research that spans a variety of scientific disciplines, from mathematics to information technology, computer science and electrical engineering. A simple definition can be found in [PP09]: “Cryptography is the science of secret writing with the goal of hiding the meaning of the message.” While this definition captures the important goal of *confidentiality*, modern cryptography aims to achieve a much wider range of goals, including data *authenticity* and *integrity*.

Differing in their key distribution mechanism and efficiency, *symmetric* and *asymmetric* ciphers together form the fundamental building blocks of modern cryptographic protocols that are utilized for secure communication.

### 2.2.1 Asymmetric Cryptography

Introduced in 1976 by W. Diffie, M. Hellmann and R. Merkle, asymmetric cryptography, or public-key cryptography, drastically simplifies key management compared to symmetric approaches [DH76]. In contrast to symmetric cryptography, each identity is assigned a key pair: a public and a private key. The public key can then be stored in an accessible way, for instance on a server, and can be utilized by any other identity to encrypt or verify messages. All of the traditional asymmetric schemes rely on one of three mathematical problems: integer factorization, discrete logarithm and the elliptic-curve discrete logarithm problem. As these problems are strongly expected to be efficiently solvable with the emergence of practical and sufficiently powerful quantum computers [Sho97], current research is fully concentrated on establishing novel post-quantum schemes that guarantee thorough security in a future age of widespread quantum computers. Asymmetric cryptography is commonly applied for modern signature schemes and key exchange protocols. Because of its limited throughput, it is not directly used for data encryption in most cases. Instead, a symmetric and shared key is derived by means of a key exchange protocol which is based on asymmetric key pairs. This combines the advantages of both symmetric and asymmetric schemes by providing efficient key management while fast data encryption is guaranteed. Such hybrid approaches are widely applied in modern complex system architectures like IoT systems or inter-vehicle communication.

### 2.2.2 Symmetric Cryptography

Symmetric ciphers are the main building blocks for guaranteeing high-throughput and confidential communication between two parties through symmetric encryption schemes. Fundamentally, a symmetric encryption scheme can be defined according to Definition 2.2.1.

**Definition 2.2.1** (Symmetric Encryption Scheme [KL14]). A symmetric (or private-key) encryption scheme is a tuple of probabilistic polynomial-time (p.p.t.) algorithms (GEN, ENC, DEC) that are defined as follows:

**GEN:** The key-generation algorithm GEN takes as an input  $1^n$  and outputs a key  $k \in \{0, 1\}^m$ , where  $m \geq n$  and  $n$  is called the security parameter.

**ENC:** The probabilistic encryption algorithm ENC takes as input the key  $k$  and a message  $m \in \{0, 1\}^*$  and outputs the cipher as  $c \leftarrow \text{ENC}_k(m)$ .

**DEC:** The algorithm DEC takes as input a ciphertext  $c$  and the key  $k$  and outputs the plaintext as  $m := \text{DEC}_k(c)$ .

The security of such a scheme is then expressed with respect to the capabilities of the adversary, i.e., the adversary model, and the security goal. Intuitively, no practical adversary should be able to retrieve any (useful) information about the plaintext by observing a fresh ciphertext. This is called *semantic security*. Practical here refers to computationally bounded adversaries. Computational security weakens the highly conservative definition of *perfect security*, which forbids the existence of *any* adversary who can gain *any* information, by restricting security to attacks that are practically feasible, i.e., that can be described by a p.p.t. algorithm, and where an adversary is allowed to retrieve negligible information about the plaintexts [KL14].

There are various threat models which can be considered when evaluating the security of an encryption scheme. For example, an adversary may be limited to observing only a single ciphertext, or he may observe a sequence of plaintext and ciphertext. Commonly considered strong security definitions are Indistinguishability Under Chosen Plaintext Attacks (IND-CPA) or even Indistinguishability Under Chosen Ciphertext Attacks (IND-CCA), where an adversary is not only able to obtain encryptions of plaintexts of his choice, but even to choose ciphertexts to be decrypted for him, thus having access to an encryption and decryption oracle. For example, IND-CPA can be formally defined as given in Definition 2.2.2.

**Definition 2.2.2** (IND-CPA [KL14]). A symmetric encryption scheme provides IND-CPA, if for all p.p.t. adversaries ADV, there is a negligible function  $\text{negl}$  such that playing the IND-CPA game described in Algorithm 1, it holds that  $\Pr[b' = b] \leq 1/2 + \text{negl}(n)$ .

---

#### Algorithm 1 IND-CPA Game

---

- (1) Key  $k$  is generated through GEN.
  - (2) The adversary ADV is given  $1^n$  and access to  $\text{ENC}_k$ . ADV constructs two plaintexts  $m_0$  and  $m_1$ .
  - (3) A bit  $b$  is chosen uniformly and at random from  $\{0, 1\}$ .  $c \leftarrow \text{ENC}_k(m_b)$  is provided to ADV.
  - (4) ADV still has oracle access to  $\text{ENC}_k$ . Eventually, ADV outputs a bit  $b'$ .
- 

Intuitively, this formally describes the inability of any adversary to learn useful information from the ciphertext alone. Note that here the randomization of ENC is crucial, because if ENC would be entirely deterministic, winning the game would be trivial as ADV has ongoing oracle access to the encryption.

### 2.2.3 Block Ciphers

The basic building blocks of symmetric encryption schemes are block ciphers, which in their theoretical sense are keyed pseudo-random permutations as defined in Definition 2.2.3.

**Definition 2.2.3** (Keyed Pseudorandom Permutation [KL14]). Let  $\pi_n$  be the set of all permutations over  $\{0, 1\}^n$  and let  $f : \{0, 1\}^n \times \{0, 1\}^a \mapsto \{0, 1\}^n$  be a polynomial-time, keyed function.  $f$  is a pseudorandom permutation if for any p.p.t. distinguisher  $D$  that either gets access to  $f_k$ , or gets access to  $F$  drawn uniformly at random from  $\pi_n$ , there is a negligible function  $\text{negl}$ , such that

$$|Pr[D^{f_k(\cdot)} = 1] - Pr[D^{F(\cdot)} = 1]| \leq \text{negl}(n),$$

where  $k$  is drawn uniformly from  $\{0, 1\}^a$ .

Intuitively, Definition 2.2.3 states that if a pseudorandom permutation is randomly keyed, it should be indistinguishable from a permutation randomly drawn from the space of all permutations of bit length  $n$ .

Block ciphers can then be instantiated in different modes of operation to form encryption schemes for arbitrary data lengths, examples being Cipher Block Chaining (CBC) and Counter (CTR) mode. It has been shown that, assuming that a cipher used is indeed a pseudorandom permutation, the encryption scheme resulting from running that cipher in CBC or CTR mode provides IND-CPA. Note that a block cipher is itself deterministic under a fixed key. The randomization is achieved by exploiting randomness in the overarching mode of operation and using random initialization vectors for each encryption (commonly, the initialization vectors are simply part of the ciphertext).

While AES is the most widely used block cipher, ciphers such as PRESENT [BKL<sup>+</sup>07] or Skinny [BJK<sup>+</sup>16] are receiving increasing attention due to their favorable area consumption in hardware. Note that while we call these designs block ciphers, there is no formal proof that they are indeed pseudorandom permutations. The scientific community is just confident that these practical ciphers are (very close to) keyed pseudorandom permutations. While this is not ideal, existing provable constructions are very inefficient and impractical [KL14].

We briefly recapitulate the theoretical foundations of symmetric encryption here, because we will see that security from a mathematical, cryptanalytic point of view is often not enough to guarantee security in practice, because the scope of the classical threat model is too limited.

### 2.2.4 Practical Block Cipher Constructions

Block ciphers operate on blocks of data with typical bit lengths of 64 or 128 bits and are encrypted with 128-, 196- or 256-bit keys. Two widely used principles to achieve the indistinguishability from randomly drawn permutations described in Definition 2.2.3 are *confusion* and *diffusion* [Sha49a]. Confusion describes the high statistical independence between the ciphertext and the plaintext, whereas diffusion describes the property that a single-bit change in the plaintext or the key should affect many bits of the ciphertext. To achieve confusion and diffusion, most modern and popular block ciphers – such as AES and PRESENT – consist of repeatedly running a substitution and permutation layer in the form of a so-called Substitution–Permutation (SP) network. Substitution is typically achieved by applying a carefully designed 4-bit or 8-bit mapping, the S-box, to the entire state of the block [KR11]. In the context of side-channel

protection, the substitution layer is the main driver of overhead, because its high nonlinearity makes it immensely challenging to mask. While there are other methodologies for constructing block ciphers, like Feistel or Lai-Massey constructions [MvOV01], we focus on SP-based constructions in this thesis.

## 2.3 Side-Channel Analysis Attacks

SCA attacks have proven to offer powerful tools to break the security guarantees of hardware systems. In the following, we will hence give an introduction to the underlying principles of such strong attacks.

### 2.3.1 Physical Attacks

The last three decades have revealed a large gap between the original black-box security definitions, as elaborated in Section 2.2.2, and the capabilities of adversaries in real-world scenarios. This is due to the fact that attackers are often not limited to a mere oracle access to the decryption or encryption algorithm where the computations themselves are entirely hidden. In practice, they are able to obtain information about the internal states of the cipher or even manipulate the execution of the algorithm in many cases. This is because cryptographic primitives are implemented either in software or in hardware on an actual physical device. By interfacing this implementation, or manipulating it, the adversary becomes much stronger than an attacker that is, for example, defined according to Definition 2.2.2. Attack scenarios where the adversary exploits the actual implementation instead of cryptanalytically analyzing the underlying algorithm are commonly referred to as *physical attacks* or *implementation attacks*. These types of attacks have proven to be highly efficient and effective in many scenarios because their requirements are often met in modern systems. As a result, they have received a great deal of attention within the research community, and a lot of effort has been devoted to thoroughly understanding these attack vectors and building strong protection mechanisms.

Physical attacks can be categorized according to [MOP07]. Here, *passive* physical attacks refer to scenarios where the cryptographic device is operated within its specification and the secret is extracted by passively observing the physical properties of the device. In contrast, *active* attacks force the device to operate outside its specification by directly manipulating the device, its input or its execution environment. A further categorization can be made according to the invasiveness of an attack. We distinguish between non-invasive, semi-invasive and invasive attacks. Side-channel attacks are non-invasive, passive attacks where only directly accessible interfaces are observed. Since these are the focus of this thesis, we will describe them in more detail.

### 2.3.2 SCA Threat Model and Security Goal

In general, for a given attack scenario, elaborating on the threat model and the security objective lays the foundation for further consideration. The threat model describes the adversaries capabilities, while the security objective describes the goal of the adversary. In the context of SCA attacks, the adversary passively monitors unintended physical characteristics of a device.

Examples of such characteristics are the imminent power consumption [KJJ99], the electromagnetic radiation [GMO01], or the timing [Koc96]. By exploiting unintended statistical dependencies between these characteristics and secret values currently stored on or processed by the device, the adversary can often completely break the security of a system, making these attacks extremely powerful and thorough protection immensely important.

**Threat Model.** Traditionally, a necessary assumption of SCA attacks was direct physical access to the attacked device. Recently, this assumption has been relaxed as completely remote scenarios have been introduced [SGMT18, ZS18, RPD<sup>+</sup>18, GKT19, OD19, SGMT21]. These attacks pose a serious security threat in shared resource architectures, such as cloud computing. Side-channel attacks can be further categorized into *profiled* and *unprofiled* scenarios. In a *profiled* attack, which is considered to be the strongest, it is assumed that the adversary has full access to a cryptographic device similar to the one under attack with power over setting the secret key. This allows him to derive a model of the device’s behavior (a template) for different keys. Now, for the actual device, an adversary can simply compare the generated templates with the actual behavior. This may allow him to efficiently retrieve the secret key. While the original model was based on a multivariate Gaussian distribution [CRR02], more complex models have been applied over time, especially with increasing advances in machine learning [MPP16, CDP17, PSB<sup>+</sup>18]. In unprofiled attacks, the adversary is limited to accessing the device under attack, but it is usually assumed that the adversary has some control over the input of the cryptographic algorithm and can trigger its execution.

**Security Goal.** Typically, the goal of an SCA attack is to recover the secret key of an implemented cipher, thereby breaking the implementation of the encryption altogether. In many cases, it is possible to recover the key directly. In other cases, SCA reduces the entropy of the key to a point where simple brute force is possible.

### 2.3.3 Power Side Channel

In this work, our focus is on power side channels. Therefore, when we talk about SCA attacks, we usually refer to power-related attack scenarios throughout this thesis.

**Power Consumption of Modern Devices.** Power SCA attacks include all attack scenarios that leverage unintentional dependencies between the data stored within or processed by a hardware circuitry and the imminent power consumption of it [MOP07]. No matter whether we consider general-purpose microcontrollers, reconfigurable hardware (FPGAs) or ASICs, data is eventually processed by a hardware circuit. For modern devices, this circuit is usually realized using Complementary Metal Oxide Semiconductor (CMOS) technology. In this context, the correlation between the processed data and the immediate power consumption stems from fluctuations in the activity of the underlying CMOS circuit. According to [MOP07], the overall power consumption of a CMOS circuit is the sum of the *static* and *dynamic* power consumption. The static power consumption refers to the consumption of the logic cells when there is no switching activity. The dynamic power consumption occurs when the input or output of logic cells switches. While for larger technology sizes ( $\geq 100$  nm), the dynamic power consumption

is the dominant driver of the overall consumption, the static power consumption becomes increasingly relevant for smaller technology sizes and both types can be leveraged for side-channel analysis [Moo22].

**Measurement.** To measure the power consumption of a cryptographic device during computation, we can measure the voltage drop over a small resistor (typically as large as  $1\Omega$  for modern technology sizes) that is placed in the GND or Vdd path of the device. This drop will be proportional to the power consumption of the device. Another option is to measure the power consumption of a (part of a) circuitry in a more localized and indirect way by measuring the EM radiation of the circuitry during computation. This is usually done by specialized EM probes that utilize a tiny coil to translate the radiation into a voltage signal. In both cases, the voltage is eventually sampled and recorded by an oscilloscope.

### 2.3.4 Experimental Leakage Assessment

From a security perspective, it is less important to certify that an implementation is resistant to a particular type of attack, but rather to assess the overall resistance to any side-channel attack. Basically, this is done by determining general statistical dependencies between observations of physical properties and sensitive data.

A common approach to evaluating practical side-channel resistance is to measure this dependency using a statistical  $t$ -test, also referred to as TVLA [GJR].

**Welch's  $t$ -Test.** The Welch  $t$ -test [Wel47] is a generalization of Student's  $t$ -test introduced in [Stu08], enabling the comparison of two populations with different variances. For two normally distributed populations  $X$  and  $Y$ , Welch's  $t$ -test is a test of the null hypothesis  $H_0$ : "The means of  $X$  and  $Y$  are equal". This can be used to indirectly identify dependencies between the internal states of the implementation and the observed characteristic, such as the power consumption of the device. For a *non-specific*  $t$ -test, the power consumption of the device is measured while processing either random or fixed plaintexts. To avoid false positives due to temporal environmental effects or state transitions between executions, it is important that the selection of whether a fixed or random plaintext is fed to the algorithm is completely random. Finally, we check whether we can distinguish the population corresponding to the fixed plaintext from the one corresponding to the measurements of the random plaintexts by their mean using Welch's  $t$ -test. The intuition is that the mean of the group of measurements with randomly supplied plaintexts converges to the average power consumption over all possible intermediate states within the implementation. Now, if we can distinguish the fixed group from this random group, there must be an internal state caused by another plaintext that we can distinguish from the state of the fixed plaintext, otherwise the means would be the same. As a result, we can indirectly identify data dependencies. Note that this only allows us to identify dependencies specific to the selected fixed plaintext, although it can often be assumed that if an implementation leaks information about the data, it does so for different fixed plaintexts.

For performing the actual  $t$ -test, we calculate the variance  $\sigma^2$  and the mean  $\mu$  for both groups of observations and calculate the  $t$ -statistic as

$$t = \frac{\mu_X - \mu_Y}{\sqrt{\frac{\sigma_X^2}{n_X} + \frac{\sigma_Y^2}{n_Y}}},$$

where  $\sigma_i^2$ ,  $\mu_i$  and  $n_i$  correspond to the variance, the mean, and the number of elements of the set of observations corresponding to population  $i \in \{X, Y\}$  respectively.

Another important parameter in hypothesis testing is the significance level  $\alpha$ , which is the tolerable probability of falsely rejecting the null hypothesis. Based on a given number of observations, and using the well-defined  $t$ -distribution, we can then translate a specified significance level into bounds for our  $t$ -statistic. If the  $t$ -value exceeds these limits and we reject the null hypothesis, the risk of being wrong would be less than our defined significance level. For the  $t$ -test, it can be shown that if we have more than 1000 observations, i.e., an estimated degree of freedom greater than 1000 – which is easily achieved in practice – a significance level lower than  $10^{-5}$  always translates into an upper bound for the absolute  $t$ -value of 4.5. In practice, we can therefore simply compute the  $t$ -statistic and assume data dependence, i.e., we assume different means of both groups, if the absolute  $t$ -value exceeds the threshold of 4.5. The risk of being wrong, i.e., of overlooking equal means, is very small, namely less than  $10^{-5}$  [SM16]. We like to emphasize that if the  $t$ -test indicates leakage, it does not mean that this leakage is exploitable by a practical attack.

**Assessment for Higher Statistical Orders.** Protected implementations often aim to avoid any data dependencies up to a certain statistical order. The test methodology of a  $t$ -test has therefore been extended to detect differences in statistical moments higher than the first standard moment, i.e., the mean [SM16]. Other statistical moments relevant in the context of SCA are the  $d^{\text{th}}$ -order centered moments and the standardized moments of a random variable  $X$  following a normal distribution, which are defined as

$$CM_d = E((X - \mu)^d), \quad SM_d = E\left(\left(\frac{X - \mu}{\sigma}\right)^d\right).$$

Conceptually, the higher-order analysis again divides the observations into two groups before calculating the mean and standard deviation. Then, each observation is preprocessed before the  $t$ -test is calculated as described above. For each observation, the mean is subtracted. For the standardized moments, the result is further divided by the standard deviation. In both cases, the resulting value is raised to the power of  $d$ . After this preprocessing, the  $t$ -test is performed as described above. Now, for a second-order analysis, the second-order centered moment  $CM_2$  is considered, which actually corresponds to comparing the variances of the two groups' distributions. For  $d \geq 3$ ,  $SM_d^2$  is considered, i.e., the higher-order standardized moments.

**Power Traces.** In practice, not only a single observation per execution is recorded. Instead, the power consumption of a device is sampled many times during one and the same execution,

---

<sup>2</sup> $SM_3$  is called the Skewness and  $SM_4$  is called the Kurtosis of a population.



i.e., the processing of a single plaintext. This results in one power *trace* over the time dimension for each execution of the considered algorithm. While the  $t$ -test can be performed in a univariate fashion, i.e., for each sample point of the trace individually, it can also be performed on combinations of sample points, using multivariate testing [SM16].

**Optimization Techniques.** Since a single trace can contain several thousand sample points, all corresponding to different power observations of a single execution of the considered algorithm, and the evaluation is often performed for many millions of traces, run time quickly becomes an issue when performing the Welch  $t$ -test as a leakage assessment. For this reason, one-pass formulas have been derived in [SM16] that allow efficient and parallel computation of the  $t$ -statistics, even at higher security orders. This makes it easy to run the leakage assessment in parallel with the power measurements since the statistics can simply be updated with each new measurement.

### 2.3.5 Correlation Power Analysis

CPA [BCO04] is a common method for extracting the secret key from a cryptographic implementation. The underlying principle is very simple and follows a divide-and-conquer methodology, where parts of the secret key can be recovered individually. More precisely, CPA is a specific form of Differential Power Analysis (DPA) [KJJ99], which generally describes power analysis attacks that lower the entropy of a secret by measuring many executions of a cryptographic implementation and applying appropriate statistical methods. While there are many different statistical methods that can be applied, like for example Mutual Information Analysis (MIA) [GBTP08] or independence testing via  $\chi^2$ -test [MRSS18, RKM19], correlation is used for CPA. To perform a CPA, we first define a power model that accurately models the power consumption of a device. This model is based on parts of the plaintext and parts of the key. Common power models are, for example, the byte-wise Hamming weight after the first substitution layer of the cipher, or the Hamming distance between neighboring bytes after the substitution layer. If we have information about the underlying implementation, it is usually much easier to find a well-fitting power model, since we can derive it by considering the internal structure and the order in which the data is processed. If we have no information about the implementation at all, finding an appropriate model usually consists of educated guesswork and a trial-and-error procedure.

As a first step, we measure the power consumption during cipher computation for many randomly selected plaintexts. In practice, for each of these plaintexts, we would measure several sampling points over time, resulting in a so-called power trace for each plaintext – similar to the  $t$ -test scenario elaborated in Section 2.3.4. Then, the following procedure is performed for each point in time separately in a univariate fashion. Let us consider a single point in time and let  $(p_i, t_i)$  be the pair of the  $i$ -th plaintext and the corresponding measured power value  $t_i$  for  $0 \leq i < n_t$  where  $n_t$  is the number of measured power values.

**Identifying Correlation.** Based on a power model  $M : \mathbb{F}_2^{|\tilde{p}|} \times \mathbb{F}_2^{|\tilde{k}|} \rightarrow \mathbb{R}$  that maps a plaintext part  $\tilde{p}$  and a key part  $\tilde{k}$  to a real value, we can derive hypothetical power values for each plaintext for which we measured the power consumption during computation. Now for each key guess, i.e., value from  $\mathbb{F}_2^{|\tilde{k}|}$ , we can compute the hypothetical power values  $h_i \leftarrow M(\tilde{p}_i, \tilde{k})$

for each  $\tilde{p}_i$  as a part of the plaintext selected from  $p_i$ . We then compute Pearson's correlation coefficient [Pea95] between the measured power values  $\mathcal{T} = \{t_i\}_{0 \leq i < n_t}$  and the hypothetical power values  $\mathcal{H} = \{h_i\}_{0 \leq i < n_t}$  as

$$\rho_{\mathcal{T}, \mathcal{H}} = \frac{\sum_{i=0}^{n_t-1} (t_i - \mu_{\mathcal{T}})(h_i - \mu_{\mathcal{H}})}{\sqrt{\sum_{i=0}^{n_t-1} (t_i - \mu_{\mathcal{T}})^2} \sqrt{\sum_{i=0}^{n_t-1} (h_i - \mu_{\mathcal{H}})^2}},$$

where  $\mu_{\mathcal{T}}$  and  $\mu_{\mathcal{H}}$  are the averages over  $\mathcal{T}$  and  $\mathcal{H}$  respectively.  $\rho$  takes values between  $-1$  and  $1$ , and is essentially an estimate of the normalized covariance between two random variables. Thus, it represents linear dependencies between two populations. The closer its absolute value is to one, the more the dependency of the populations can be described by a linear relationship. Now, assuming that our model  $M$  accurately models the power consumption of the attacked implementation, we expect to see the highest correlation between  $\mathcal{T}$  and  $\mathcal{H}$  for the correct key guess  $\tilde{k}$ . While we are only using classical CPA in the course of this thesis, there are also techniques for performing higher-order attacks by means of the correlation. Furthermore, similar to performing the  $t$ -test statistics, one-pass formulas for efficiently performing CPA are introduced in [SMG16].

**Selection of the Power Model.** To avoid false-positive key guesses, we typically choose a power model in which the hypothetical power values for a false key guess appear mostly random compared to the real power consumption, and thus their relation cannot be well approximated by a linear function. To achieve this, we can actually use the general structure of a cipher to our advantage by choosing a power value based on intermediate values after the substitution layer. As described in Section 2.2.4, an S-Box is designed to guarantee that a single bit flip in the input will affect many bits in the output. Thus, even a wrong key guess with a single bit difference to the correct one will result in an output distribution that is far from the actual one, and thus correlates very poorly with the actual power consumption.

For an AES implementation, common power models include  $M(\tilde{p}, \tilde{k}) = \text{HW}(\text{S}(\tilde{p} \oplus \tilde{k}))$  or  $M(\tilde{p}' || \tilde{p}'', \tilde{k}' || \tilde{k}'') = \text{HD}(\text{S}(\tilde{p}' \oplus \tilde{k}'), \text{S}(\tilde{p}'' \oplus \tilde{k}''))$ , where  $\text{HW}$  corresponds to the Hamming weight of a value,  $\text{HD}$  is the Hamming distance between two values and  $\text{S}$  denotes the S-box Mapping;  $\tilde{p}$  and  $\tilde{k}$  are bytes selected from the plaintext and from the key respectively. With this methodology, we can recover the key part by part, effectively reducing the complexity compared to traditional cryptanalytical key recovery attacks. If, for example,  $M(\tilde{p}, \tilde{k}) = \text{HW}(\text{S}(\tilde{p} \oplus \tilde{k}))$  is an accurate model for an AES-128 implementation, we can reduce the complexity to  $16 \cdot 2^8 = 2^{12}$  by guessing the key byte-wise and identifying the right key byte through CPA.

## 2.4 SCA Countermeasures

There are several approaches to counteract side-channel leakage, which can be broadly categorized into *hiding* and *masking* according to [MOP07]. Since SCA exploits dependencies between intermediate values processed by a cryptographic device and physical properties, countermeasures naturally aim to weaken this dependency.

### 2.4.1 Hiding Countermeasure

Unlike the masking countermeasure, the hiding countermeasure aims to reduce this dependency without changing the processed data, i.e., without changing the intermediate values [MOP07]. Hiding can be applied in both the time and the amplitude domain. Hiding in the time domain aims at randomly changing the time of the operations performed. In hardware, this can be achieved by randomly changing the clock frequency or by performing dummy operations during random dummy cycles on random data. In this way, the relevant information within a power trace will be at different sample points for each execution of the implementation. Hiding in the amplitude domain involves either reducing the signal, i.e., the useful information contained in the measured observations, or increasing the noise within an operation. Reducing the signal can be achieved by equalizing the power consumption. A well-studied approach in this context is the use of Dual-Rail Precharge (DRP) logic [TAV02, TV04]. Noise can be increased by implementing a dedicated noise engine that performs noisy operations on random data during each clock cycle.

### 2.4.2 Masking Countermeasure

Ultimately, masking aims to make it infeasible for any realistic adversary to obtain secret information from side-channel leakage observations by randomizing the processed data. For symmetric ciphers – which we focus on in this thesis – Boolean masking is commonly used. According to Definition 2.4.1, the goal of Boolean masking is to split a secret into independent random shares so that information about the secret is revealed only if information about all of them is recovered.

**Definition 2.4.1** (Boolean Masking [CJRR99]). Based on the idea of secret sharing [Sha79], *Boolean Masking* splits a sensitive value  $X \in \mathbb{F}_2$  into  $s \geq 2$  independent random shares  $X_i \in \mathbb{F}_2$  such that  $X = \bigoplus_{i=0}^{s-1} X_i$ . Then, any combination of up to  $s - 1$  shares does not reveal anything about  $X$ .

**Definition 2.4.2** (Uniform Masking [Bil15]). A masking  $\mathbf{X} = (X_0, X_1, \dots, X_{s-1})$  of a secret  $X \in \mathbb{F}_2$  is uniform if and only if it is uniformly distributed over all valid sharings of  $X$ , i.e., if and only if for all  $x$  it holds that

$$Pr[\mathbf{X} = \mathbf{x} \mid X = x] = \begin{cases} p, & \text{if } \mathbf{x} \in Sh(x), \\ 0, & \text{otherwise,} \end{cases}$$

and

$$\sum_{\mathbf{x} \in Sh(x)} Pr[\mathbf{X} = \mathbf{x}] = Pr[X = x].$$

Usually, Boolean masking of a secret value  $x$  is practically achieved by drawing  $s - 1$  shares uniformly at random from  $\mathbb{F}_2$  and calculating the remaining share as the secret XORed with all other shares. Note that through this procedure, we achieve uniform masking as defined in Definition 2.4.2. This ensures that no information with respect to the unshared value can be retrieved from any proper subset of the shares.

The intuition behind masking in hardware becomes clear when considering a simple example. Suppose that the power consumption of a set of wires is equal to the sum of the bit values

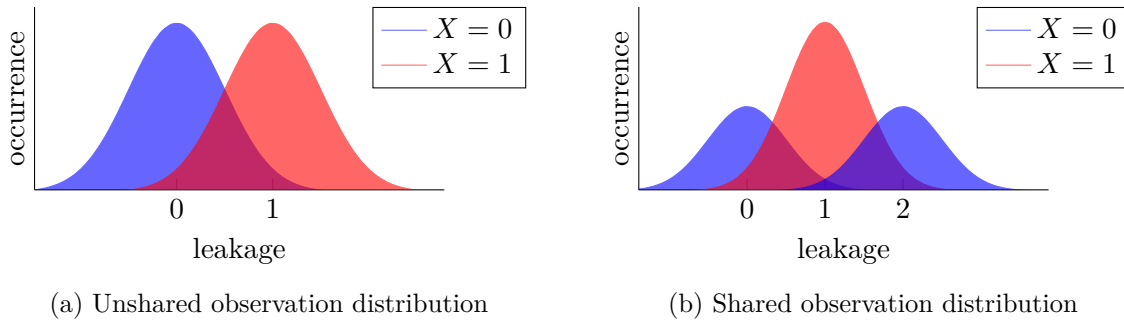


Figure 2.1: Comparison unshared vs. shared observation distribution

carried by each of the wires plus a normally distributed noise. For simplicity, we further assume a single unshared input bit which is uniformly distributed. Under these assumptions, without sharing the sensitive bit value, we can simply distinguish between the two possible values 0 and 1 by looking at the mean of the power observations. This is shown schematically in Figure 2.1a. If we now randomly split the sensitive variable into two parts, we would observe a distribution similar to the one shown in Figure 2.1b. The value would be split into  $(X_0, X_1) = (0, 1)$  or  $(1, 0)$  with the sum of the wires equal to one if the value is one. If the unshared value is zero, the sharing is  $(X_0, X_1) = (0, 0)$  or  $(1, 1)$  and the sum is either zero or two. Now it becomes clear that the mean of the observations is the same for both nonshared cases 0 and 1, so it is not possible to distinguish the underlying secret by the mean of the observations. By implementing masking, we force the adversary to consider higher statistical moments [BDF<sup>+</sup>16], such as the variance as a second-order centralized moment. For sufficient noise levels, it can be shown that the complexity of such distinguishing grows exponentially with the security order [BDF<sup>+</sup>16]. As a result, masking provides a powerful tool to push the complexity for successful SCA attacks into the practical impossible.

## 2.5 Formal SCA Adversary Models

To capture and formalize the capabilities of an adversary in the context of SCA, abstractions of a circuit, physical effects and probing capabilities had to be derived. In the following, we give an overview of these abstractions.

### 2.5.1 Circuit Model

In accordance to [AIS18], we use the *Encoded Circuit Model* to model the behavior of a hardware circuit in an abstract way.

As considered in [ISW03] and later extended in [CS21], any stateful and deterministic circuit  $C$  is modeled as a Directed Acyclic Graph (DAG)  $G_C = \{\mathcal{V}, \mathcal{E}\}$ , where  $\mathcal{V}$  is the set of vertices and  $\mathcal{E}$  is the set of edges in  $G_C$ . In this context, edges are the wires carrying elements from  $\mathbb{F}_2$ , while vertices are either initial inputs to the combinatorial circuit, logic gates such as AND and XOR, refresh gates, or memory gates (flip-flops). On each invocation of the circuit (each rising edge of the clock signal) a refresh gate, which has no inputs, outputs a freshly drawn random bit by tossing a fair coin. Memory gates work by outputting the previous input to the

gate while storing the next input for the next invocation of the circuit (the next clock edge). Overall, the circuit implements a Boolean function  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$  – while possibly requiring multiple invocations.

Because it is based on a DAG structure, this model lacks the ability for the circuit to contain feedback loops, which occur in many compact and iterative designs. As a result, it has been extended in [CS21] to allow modeling of multiple executions of the same physical module. Essentially, the definition is extended to a simple directed graph, where nodes correspond to states of physical gates during different clock cycles. Here, one and the same physical gate is called a *structural gate* and a physical wire connecting two structural gates is called a *structural wire*. A circuit execution is then reflected by a graph corresponding to all state transitions of structural gates over time, as defined in Definition 2.5.1.

**Definition 2.5.1** (Circuit Execution [CS21]). A circuit execution of a structural circuit  $G_C = (\mathcal{G}, \mathcal{W})$  for the set of cycles  $\mathcal{T}$  is a directed graph  $G'_C = (\mathcal{V}, \mathcal{E})$  with  $\mathcal{V} \in \mathcal{G} \times \mathcal{T}$  and  $\mathcal{E} \in \mathcal{W} \times \mathcal{T}$  where wires connect the gates according to their latency. Here,  $\mathcal{G}$  and  $\mathcal{W}$  denote the structural gates and wires of  $C$ .

These abstractions provide all the necessary means to abstract the behavior of a hardware circuit and to define adversary capabilities for SCA attacks.

To limit adversarial probing to the masked circuit, and to exclude masking and unmasking of input data from being observed, the secure computation process is divided into three steps. As defined in [AIS18], a *circuit compiler* is defined by a tuple of three algorithms (COMPILE, ENCODE, DECODE) which are defined as follows.

- The COMPILE algorithm is deterministic and takes as input a structural circuit  $C$  and outputs a randomized (masked) circuit  $\tilde{C}$ .
- ENCODE is a probabilistic algorithm that takes as input the unshared input  $X$  to  $C$  and outputs the encoded input  $\mathbf{X}$  to  $\tilde{C}$ . This encoded input corresponds to the shared representation of the data, which – in our case – is derived by means of Boolean masking.
- Eventually, DECODE is a deterministic algorithm that takes the encoded data  $\mathbf{Y}$  and decodes/unshares it to achieve the unshared output  $Y$  of  $C$ .

As a result, these algorithms allow sharing the initial input data (via ENCODE), computing on the shared representation of the input (via  $\tilde{C}$ ), and unsharing the result (via DECODE), so that  $Y \leftarrow (\text{DECODE} \circ \tilde{C} \circ \text{ENCODE})(X)$  for  $Y \leftarrow C(X)$ , while the adversary is restricted to making observations only within  $\tilde{C}$ . This means that the algorithms ENCODE and DECODE are known to the adversary, but the actual execution cannot be probed.

## 2.5.2 Noisy Leakage Model

The noisy leakage model – introduced in 1999 – is assumed to be very close to the actual physical leakage processes. Its drawback is its low level of abstraction, which makes it very involved to work with. As a result, formulating and deriving sound security arguments within this model relies on complex information-theoretical arguments.

According to Chari et al. [CJRR99], each wire of the circuit leaks its value as described by  $\text{noise}(X) \leftarrow X + \chi$ , where  $X$  describes the value of the wire and  $\chi$  follows a normal distribution.

The security, or SCA resilience, of a circuit is then defined by the number of observation traces, i.e., the observed executions of the circuit required to obtain the processed secret.

The original noisy leakage model was later generalized in [PR13], which describes a more general noise function  $\text{noise}$  defined by the statistical distance – denoted by  $\delta$  – between a uniform  $X$  and  $X|\text{noise}(X)$ . Intuitively, the statistical distance describes how much information of  $X$  is contained in the noisy leakage observations  $\text{noise}(X)$ . This has the advantage that it generalizes the noise function from considering only additive Gaussian noise to include more complex noise functions.

This model is well suited to explain why masking is such an effective countermeasure against SCA attacks. In [CJRR99], it was shown that, given sufficient noise, identifying useful information from noisy shares becomes exponentially harder as the number of shares increases. Masking can thus be viewed as noise amplification with respect to the unshared value. Note that in theory, security against SCA attacks can be achieved without any masking, simply by having an immense amount of noise in each observation, so that deriving useful information becomes impossible for any realistic adversary. However, achieving such a high noise level is not really feasible in practice, and it is therefore advantageous to implement a hybrid approach that increases the noise contained in each observation by technical means and further amplifies the noise with respect to the unshared secret by masking on an algorithmic level.

### 2.5.3 Random Probing Model

While conceptually very close, the  $\epsilon$ -random probing model further increases the abstraction compared to the noisy leakage model. It uses a very simple noisy leakage function for each wire, namely

$$\varphi(X) = \begin{cases} X & \text{with probability } \epsilon, \\ \perp, & \text{otherwise,} \end{cases} \quad (2.1)$$

where  $\perp$  symbolizes the null value, meaning nothing is leaked. For each execution, each wire hence leaks its value with probability  $\epsilon$ . This model was later relaxed by Dziembowski et al. in [DFS15] in the form of the average  $\epsilon$ -random probing model that allows for a tighter reduction to the noisy leakage model. This was achieved by a seemingly small twist: Equation 2.1 does not need to hold for all possible wire values, but only on average over all possible values for  $X$  (note that the definition for the model does not restrict wires to carrying elements from  $\mathbb{F}_2$ ).

For a single execution of a circuit, the observation is given by a set of wires and their corresponding values that have been leaked according to the simple leakage function  $\varphi$ . The security level in the random probing model is then simply given by the probability that an observation under a defined noise level  $\epsilon$  leaks information about the secret, i.e., depends on the secret. This is a very intuitive security definition, because it gives us the expected number of traces until an adversary receives a leaking set. In practice, this is computed by counting all leaking probe sets for all possible set sizes, and then computing the probability with respect to a Bernoulli distribution.

### 2.5.4 $d$ -Probing Model

Because of its high level of abstraction, most security arguments in the area of SCA resistance are settled in the  $d$ -probing adversary model [ISW03]. Here, an adversary is granted the ability

---

**Algorithm 2** glitch-extend

---

**Input:** Probe  $P$ **Output:** Glitch Extension of  $P$ :  $\mathcal{P}_{ext}$ 

```

if  $P$  is placed on an output of a combinatorial gate then
     $\mathcal{P}_{ext} \leftarrow \bigcup_{0 \leq i < n} \text{glitch-extend}(P_i)$   $\triangleright$  where  $P_i$ ,  $0 \leq i < n$  are probes placed onto all inputs to
    the combinatorial gate
else
    if  $P$  is placed on an output of a register or on a primary input then
         $\mathcal{P}_{ext} \leftarrow \{P\}$ 
    end if
end if

```

---

to place up to  $d$  probes on the wires of the circuit and should not learn any information about the processed secret. More precisely, security in the  $d$ -probing adversary model is given if the distribution over the observations measured by the probes is independent of the distribution of the processed secret.

The standard model was later extended by Faust et al. [FGP<sup>+</sup>18] in form of the *robust*  $d$ -probing model to cover leakage-enhancing physical effects that occur in the hardware, such as *glitches*, *transitions* at register stages, and *coupling* between wires. For each physical effect, the set of probes is extended in a specific way, with the goal of always covering the worst-case scenario, i.e., the maximum amount of information an adversary can observe. Fundamentally, the robust  $d$ -probing model covers the fact that the leakages of wires within a circuit cannot be considered independently in many cases.

**Glitches.** In the SCA context, glitches are unintentional signal recombinations caused by different path delays between two register stages. These recombinations may allow an adversary to observe not only the stable and settled signal on a wire, but possibly other inputs of the combinatorial circuit on which a probe is placed. Thus, when glitches are considered in the context of the robust  $d$ -probing adversary model, a probe placed on a wire of a circuit is extended to a new set of probes placed on each stable signal that contributes to the wire's computationally intended value. These signals are either register outputs or primary inputs to the circuit. A pseudo-algorithm for deriving such a probe extension for a single probe is given in Algorithm 2. When extending a set of probes, each probe can be extended separately before the union of the extensions is built. A glitch extension of a probe set  $\{P, Q\}$  on an example circuit can be seen in Figure 2.2a. The initial set of probes is essentially replaced by a stronger set of probes  $\{P_1, P_2, P_3, P_4\}$  and security is evaluated under this new set of probes, i.e., the independence of the observations and the secret is checked.

**Transitions.** Data *transitions* on registers (flip-flops) are known to be an important source of leakage information in hardware circuits. When considering transitions in the robust probing model, a probe placed on a register can always observe the values of two consecutive clock cycles, so that a single-valued observation is extended to an observation tuple of size two. An illustration of a set of probes that is glitch- and transition-extended is given in Figure 2.2b.

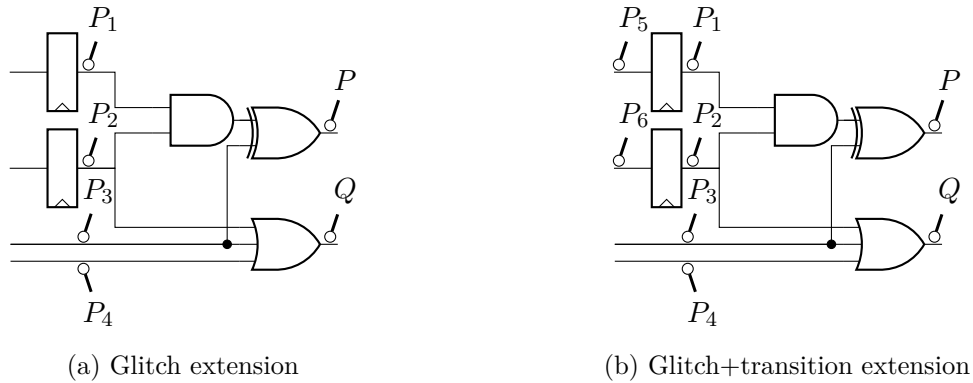


Figure 2.2: Probe extension on exemplary circuit

Again, the security check is simply preceded by a preprocessing step that effectively replaces the original probe set with a stronger one (in this case  $\{P_0, P_1, \dots, P_6\}$ ).

**Coupling.** *Coupling* between wires describes signal recombinations within a circuit caused by the close proximity of adjacent wires. Here, when a probe is placed on a wire, it can be extended to observe adjacent wires. As a result, the original probe is replaced by a set of probes containing all probes on wires within a certain radius (including itself).

**The  $(g,t,c)$ -Robust Probing Model.** A tuple  $(g, t, c)$  indicates the physical effects considered by the robust probing model. Here,  $g, t \in \{0, 1\}$  indicate whether glitches or transitions are considered, and  $c \geq 0$  describes how many adjacent wires are considered for coupling. For example, security under the  $(1, 1, 0)$ -extended  $d$ -probing model is given if all considered probes are glitch- and transition-extended before checking for security similar to the standard probing model, but considering the new set of probes. The robust probing model is thus a natural extension of the standard probing model, which hands the adversary an extended set of probes.

**Model Used in This Thesis.** We use the robust  $d$ -probing model as a formal SCA adversary model throughout this thesis. The reason for this is twofold. Both the noisy leakage model and the random probing model assume independent leakage of wires, and thus do not account for recombining physical effects in hardware that can degrade security. Furthermore, it has the highest level of abstraction, which makes it very convenient to work with.

### 2.5.5 Bounded Moment Model

The bounded moment model, introduced by Barthe et al. in [BDF<sup>+</sup>16], provides an intuitive explanation of why security with respect to the  $d$ -probing model usually results in high practical security evaluated by common TVLA methods. In this model, the leakage  $L_c$  during a single clock cycle of a hardware circuit execution is assumed to be the arithmetic sum of all values processed during that cycle. Considering the arithmetic sum is crucial as it maintains the independence assumption, i.e., the values are not combined in a nonlinear fashion.



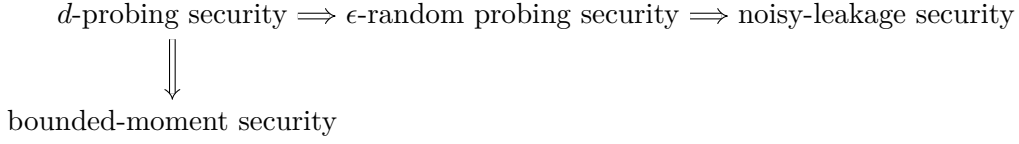


Figure 2.3: Security reductions between formal SCA adversary models

**Definition 2.5.2** (Mixed Moment [BDF<sup>+</sup>16]). Given a set of random variables  $\{X_i\}_{i=1}^r$  and positive integers  $o_1, o_2, \dots, o_r$ , the expectation

$$E(X_1^{o_1} \cdot X_2^{o_2} \cdot \dots \cdot X_r^{o_r})$$

is called mixed moment of order  $o = \sum_i o_i$ .

Given Definition 2.5.2, security in the bounded moment model under order  $o$  is given if and only if all the mixed moments up to order  $o$  of  $\{L_c\}_{c=1}^{n_c}$  are independent of the secret. Here,  $\{L_c\}_{c=1}^{n_c}$  is the set of leakages over all  $n_c$  cycles of an implementation’s execution. It was further shown in [BDF<sup>+</sup>16] that security in the  $d$ -probing model implies security in the bounded moment model under  $o = d$ , meaning we can construct a  $d$ -probing adversary from an adversary in the bounded moment model.

In the bounded moment model, the security of an implementation is expressed as the independence of specific statistical moments of the observations and the processed secret. This is analogous to what is commonly tested through a  $t$ -test methodology for leakage assessment. As a result, the reduction between the bounded moment model and the  $d$ -probing model links security in the  $d$ -probing model to security under common experimental TVLA.

It further offers an explanation for the effectiveness of masking, as identifying dependencies between the secret and a moment of an observation becomes exponentially hard with increasing order  $o$  if the leakage is sufficiently noisy.

### 2.5.6 Relations Between Formal Adversary Models

Figure 2.3 provides an overview of the relations between the formal SCA adversary models described above. The arrow indicates that there is a security reduction between the two models. This means that an adversary in the  $d$ -probing model can be constructed from an adversary in the noisy leakage model. By inverting this argument,  $d$ -probing security implies security in the noisy leakage model. In the following, we give an intuition of how these reductions work. For a formal elaboration, we refer the interested reader to [DDF19] and [BDF<sup>+</sup>16].

**From Noisy Leakage to Random Probing.** To construct an adversary in the  $\epsilon$ -random probing model from an adversary in the noisy leakage model, we can show how to derive a leakage simulator in the random probing model that offers the same information to an adversary as the noisy leakage. Let us consider the noisy leakage over a wire carrying values of  $\mathbb{F}_2$ . The noise here describes how much information about a processed value  $X \in \mathbb{F}_2$  can be derived by observing the noisy value  $\text{noise}(X)$  and is identified by the statistical distance  $\delta$  – as described in Section 2.5.2. The two extreme cases are depicted in Figure 2.4. In Figure 2.4a, we have an example distribution over a wire with no noise, as we have very high (nearly complete) certainty

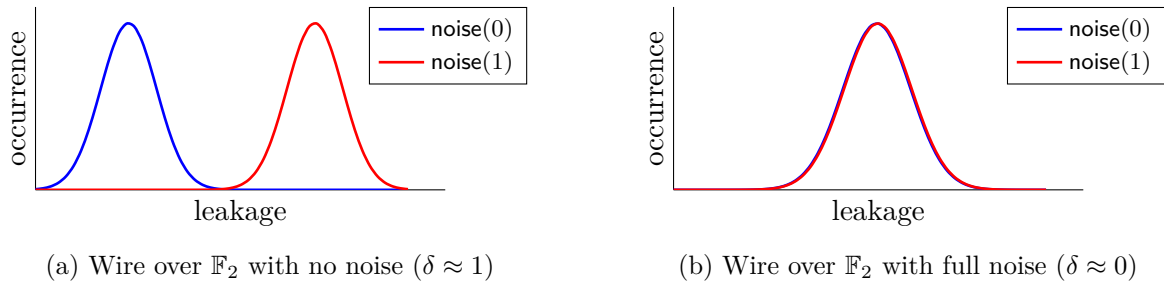


Figure 2.4: Low noise vs. high noise in the noisy leakage model

about  $X$  when observing  $\text{noise}(X)$ . This can be trivially modeled by a leakage simulator in the  $\epsilon$ -random probing model by setting  $\epsilon = 1$ , i.e.,  $\varphi(X) = X$  and an adversary would simply receive the value of that wire with each query. In Figure 2.4b, a distribution with full noise is shown. By observing the noisy value, we actually gain no certainty about the value whatsoever. This again can trivially be modeled by setting  $\varphi(X) = \perp$ , i.e.,  $\epsilon = 0$  in the random probing model. Following this intuition, it was shown that an  $\epsilon$ -random probing adversary can be constructed from any  $\delta$ -noisy adversary [DDF19] and that random-probing security implies noisy-leakage security.

**From Random Probing to d-Probing.** Now we can construct a leakage simulator that models the behavior in the  $d$ -probing model based on an adversary in the random probing model. This is done more or less by directly applying Chernoff's bound [Che52], since leakage in the random probing model is modeled in a binominal fashion [DDF19]. This allows us to derive an upper bound on the probability that more than a certain number of wires will leak. As a result, we can construct a  $d$ -probing adversary from an adversary in the random probing model for a specific  $d$  depending on the noise level of the circuit.

**From Bounded Moment to d-Probing.** In the bounded moment model, leakage is described by the noisy arithmetic sum of all values processed during a single clock cycle. Let us consider a circuit with wires carrying elements from  $\mathbb{F}_2$ , where we have two clock cycles, each processing three values. The noiseless leakage during the cycles can be described by  $L_1 = Y_1 + Y_2 + Y_3$  and  $L_2 = Z_1 + Z_2 + Z_3$ . Second-order security in the bounded moment model is given if  $E(L_1)$ ,  $E(L_2)$ ,  $E(L_1 \cdot L_2)$ ,  $E((L_1)^2)$ , and  $E((L_2)^2)$  are each independent of the processed secret. Due to the linearity of expectation, all of these can be described by sums of different expectations of the form  $E(Y_1^{o_1} \cdot Y_2^{o_2} \cdot Y_3^{o_3} \cdot Z_1^{o_4} \cdot Z_2^{o_5} \cdot Z_3^{o_6})$ , where  $\sum_{i=1}^6 o_i \leq 2 = d$ , i.e., each of these terms contains only expectations of monomials consisting of at most  $d = 2$  values. The independence of all tuples containing a maximum of two of the processed variables from the secret implies the independence of these terms. This is exactly what is checked in the  $d$ -probing model.

## 2.6 Automated Hardware Masking and Composability Notions

When it comes to masking, a wide spectrum of approaches for creating a masked hardware implementation exists. At one end of the spectrum, there is the *handcrafted masking* approach

which mainly relies on heuristics. At the opposite end, there is the fully *automated masking* approach, which is grounded in a strong theoretical foundation and offers robust security guarantees in the face of formal adversary models. Each approach has its unique set of benefits and drawbacks, and a multitude of hybrid strategies aim to harness the strengths of both methodologies.

### 2.6.1 Handcrafted Masking and Experimental Evaluation

**Handcrafted Masking.** For a certain period of time, masking a circuit was mainly based on heuristics. A common approach to achieving side-channel resistance was to divide a circuit into different functional blocks before heuristically masking those blocks, and finally arguing why the overarching, interconnected design is secure. Achieving solid masking – such as masking that is consistent with the robust  $d$ -probing model – through such handcrafting is a difficult and time-consuming task that is highly error-prone, even for experts with several years of experience in side-channel-resistant cipher design. Furthermore, the process of handcrafted masking for a particular cipher and architecture (round-based, serialized, unrolled) is not directly applicable to other ciphers and architectures. For each specific design, the masking process has to be done more or less from scratch, involving highly trained experts and consuming a high amount of resources and time. Furthermore, seemingly minor errors can render an implementation completely insecure in practice. Various examples for designs that were carefully, but manually, designed can be found in the literature [Sug19, SBHM20, SBM21, SM21].

**Experimental Evaluation.** Traditionally, masked implementations were validated solely by experimental analysis, specifically practical TVLA, without any proof or verification that they conformed to a formal security definition. With this approach, the security statement regarding the SCA resilience of an implementation is always bound to the experimental context, i.e., the setup and environment. Running the implementation in a different context, e.g., on a different technology or platform, or with different experimental equipment, may affect its security. Formal adversary models, such as the  $d$ -probing model, abstract from such conditions by conservatively modeling the leakage behavior of a cryptographic circuit, and allow a more general security statement to be made. As a consequence, a relevant part of the research community has started focusing on systematic ways of applying masking to unprotected implementations that result in security guarantees with respect to a formal adversary model. Due to its high level of abstraction, most of these schemes are analyzed with respect to the robust  $d$ -probing adversary as described in Section 2.5.4.

### 2.6.2 Towards Sound and Systematic Hardware Masking

As conformity to the (robust)  $d$ -probing model usually results in high practical security, part of the research focus has transitioned from manually designing masking schemes and experimentally verifying the resulting post-silicon implementation to systematic approaches that result in provably secure implementations with respect to the formal (robust)  $d$ -probing adversary model.

A first attempt to realize provable security in the presence of glitches is the so-called Threshold Implementation (TI) [NRR06]. Here, a function is implemented as a vector of  $s_{out}$  component functions  $\mathbf{f} = (f_0, f_1, \dots, f_{s_{out}-1})$  realized as combinatorial circuits computing the output shares.

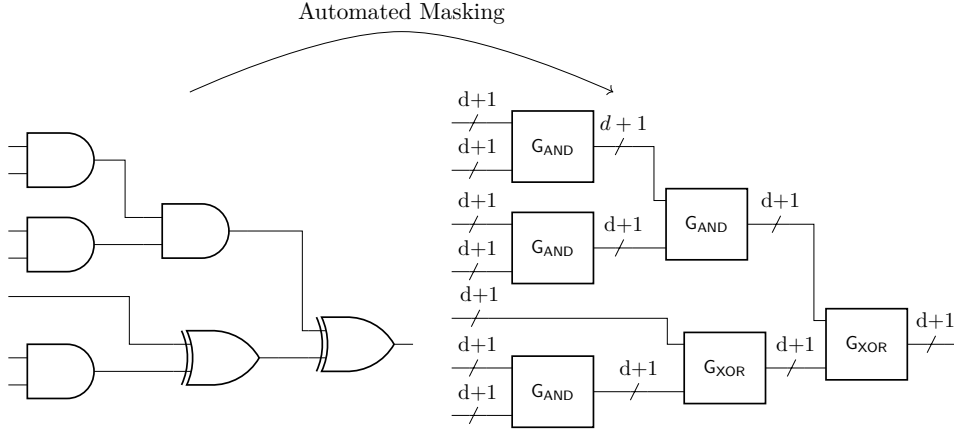


Figure 2.5: Automated Masking Concept

Besides its correct functionality, the concept of TIs is rooted in two main security-related properties of a shared circuit:

**Non-Completeness.** Any combination of up to  $d$  component functions  $f_i$  must be independent of at least one input share per input.

**Output Uniformity.** The output of the shared function must be a uniform masking of the unshared function output as defined in Definition 2.4.2. This is equivalent to saying that the TI must be uniform according to Definition 2.6.1.

**Definition 2.6.1** (Uniform sharing of a function [Bil15]). Let  $s_{in}$  be the number of shares per input. The  $d^{th}$ -order sharing  $\mathbf{f}$  of a circuit operating on  $\mathbb{F}_2$  and realizing  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$  is uniform if for all  $x \in \mathbb{F}_2^n$  and  $y \in \mathbb{F}_2^m$  with  $f(x) = y$ , and all  $\mathbf{y} \in Sh(y)$  and  $s_{out} \geq d + 1$ , it holds that

$$|\{\mathbf{x} \in Sh(x) | \mathbf{f}(\mathbf{x}) = \mathbf{y}\}| = \frac{2^{n(s_{in}-1)}}{2^{m(s_{out}-1)}}.$$

Intuitively, the non-completeness property ensures that placing up to  $d$  probes on the circuit will only reveal a part of all shares and never all of them – even in the presence of glitches. Then, if the input sharing is uniform according to Definition 2.4.2, the observations will be completely independent of any secret. The output uniformity property aims to ensure composability, meaning that the input to another TI will again constitute a uniform sharing. Elaborations on higher security orders were given by Bilgin et al. in [BGN<sup>+</sup>14], before it was shown that the TI properties are not sufficient for guaranteeing security in the higher-order  $d$ -probing model in [RBN<sup>+</sup>15]. Also, for each new composition of TI constructions, the output uniformity has to be re-evaluated. Other systematic approaches were introduced in [RBN<sup>+</sup>15], [GIB18] and [GM17]. Later, it was shown that all of these schemes suffer from higher-order design and composability flaws when initialized in hardware [MMSS19]. This is why part of the focus was laid on finding a modular masking approach that is automatable and less prone to security flaws.

### 2.6.3 Gadget-Based Masking

Gadget-based masking is founded on a modular divide-and-conquer approach. Instead of masking the cipher on a global level, it allows building a protected circuit from small building blocks that individually satisfy specific and well-defined properties that guarantee that the composed circuit is secure in the (robust)  $d$ -probing model. These building blocks are usually masked variants of small logic gates such as AND or XOR [BBD<sup>+</sup>16, CS20, CGLS21, KM22b, KM22a], but building blocks with larger functionality also exist and are introduced in this thesis [KSM22]. We refer to these composable and masked hardware modules as *gadgets*. Most gadgets with atomic functionality can be instantiated at any security level, allowing the unprotected circuit to be systematically masked for arbitrary security levels, thereby tailoring the level of protection to a specific use case. The basic principle of gadget-based masking is straightforward and is illustrated schematically in Figure 2.5, where the unprotected combinatorial circuit is simply transformed by inserting appropriate composable gadgets. Here, only gadgets with atomic gate functionality were used. Gadget-based masking has significant advantages over heuristic masking. Due to its modular approach based on components that are designed to individually satisfy well-defined theoretical properties that guarantee secure composability, a circuit composed of gadgets is provably secure in the formal robust  $d$ -probing adversary model and thus also provides strong security guarantees with respect to practical SCA resistance as elaborated in Section 2.5. Because of its theoretical and modular foundation, this approach is well suited to be fully automated, eliminating the need for additional extensive post-masking security verification. Complete automation reduces the need for expertise in the area of hardware masking, while providing an easy way to produce hardware that is thoroughly protected against SCA attacks. Furthermore, it can be easily applied to any symmetric cryptographic primitive or cipher, implemented in any architectural fashion, such as serialized, round-based, or fully unrolled.

Rooted in its generic applicability, gadget-based masking has some relevant drawbacks compared to heuristic masking tailored to a specific design, i.e., algorithm and architecture. In addition to processing multiple signals (shares) per unshared input, each gadget introduces individual overhead into the design in form of requiring register stages and fresh random bits. This results in a higher overhead introduced in terms of area, randomness requirements and latency of the overarching composed circuit, especially when compared to heuristically masked designs.

Due to its many advantages, it is highly beneficial to find optimized constructions for gadget-based masking that reduce overhead requirements, or to find a good trade-off between latency, randomness requirements and area overhead for different use cases and applications.

### 2.6.4 Composability Notions

The theoretical foundation for achieving fully automatic masking that guarantees security in the (robust)  $d$ -probing model is laid by so-called *composability notions*. Ultimately, these notions define gadget-individual properties that aim to define sufficient conditions for ensuring the secure interconnection of gadgets to form larger cryptographic (or any digital) circuits with strong guarantees with respect to SCA resilience.

**Probe Simulation.** To better describe the essence of these notions, we define *perfect probe simulation* in Definition 2.6.2. For each possible probe placement, it must be possible to con-

struct a simulator according to this definition such that the simulator only works on a proper subset of all shares of each input and not on all of them. The exact restriction with respect to the set of input shares is defined by the composability notion.

Fundamentally, for a given set of probes  $\mathcal{P} = \{P_0, P_1, \dots, P_p\}$  and when fixing all shares in the simulation set  $\mathcal{S}$ , the output of the constructed simulator has to follow the same distribution (is described by the same probability mass function) as the probes placed on the real circuit, i.e., the distribution over  $(P_0, P_1, \dots, P_p)$ , only by using  $\mathcal{S}$  and without any knowledge of the distribution over other inputs to the masked circuit. This must hold for every possible, fixed value assignment to  $\mathcal{S}$ . In this context, it is crucial to differentiate internal randomness from inputs to the circuit, although they are supplied by an external randomness source in practice.

To better understand what the concept of perfect probe simulation describes, we can consider the two simple examples presented in Figure 2.6. In Figure 2.6a, a single probe is placed on the registered output of an AND gate that takes both shares of a secret as input. Now in order to perfectly simulate the probe, both shares are required. Without loss of generality, we assume that  $X_0 = 1$ . In this case the distribution over  $X_1$  is forwarded to the output. Without knowledge of  $X_1$ , we cannot simulate the output of the AND gate. Hence, we need both of them to simulate the output for all possible values of  $(X_0, X_1)$ . The same is true if we replace the AND gate by an XOR gate. The situation changes if we consider Figure 2.6b. Here we refresh one of the shares before the AND operation is performed. The refresh is assumed to be done by internal randomness, meaning we can assume that the blinding bit is drawn from a uniform distribution over  $\mathbb{F}_2$ . We also depict the observation distribution for  $X_0 = 0$  and  $X_0 = 1$ , respectively. Because of the blinding, the output of the XOR gate will follow a uniform distribution over  $\mathbb{F}_2$ , i.e., it is derived by a fair coin toss, independent of  $X_1$ . Hence, if  $X_0 = 1$ , this distribution is forwarded to the output. We can simulate this by simply tossing a coin. If  $X_0 = 0$ , the output will always be 0 due to the nature of an AND gate. A simulator `sim` can hence be simply build according to Algorithm 3.

**Definition 2.6.2** (Perfect Probe Simulation). Let  $\mathcal{P}$  be a set of probes placed on a masked circuit. Then  $\mathcal{P}$  is *perfectly simulatable* by a set  $\mathcal{S}$  of input shares if and only if there exists a p.p.t. simulator `sim` such that for any values of the inputs to the masked circuit, the probability distribution over  $\mathcal{P}$  and `sim`( $\mathcal{S}$ ) are equal.

**Probe Propagation.** The idea of a probe simulator was later picked up in [CS20], where the related concept of *probe propagation* was introduced. Here, a set of probes is said to propagate

---

**Algorithm 3** `sim`: A simulator to simulate  $P$  in Figure 2.6b

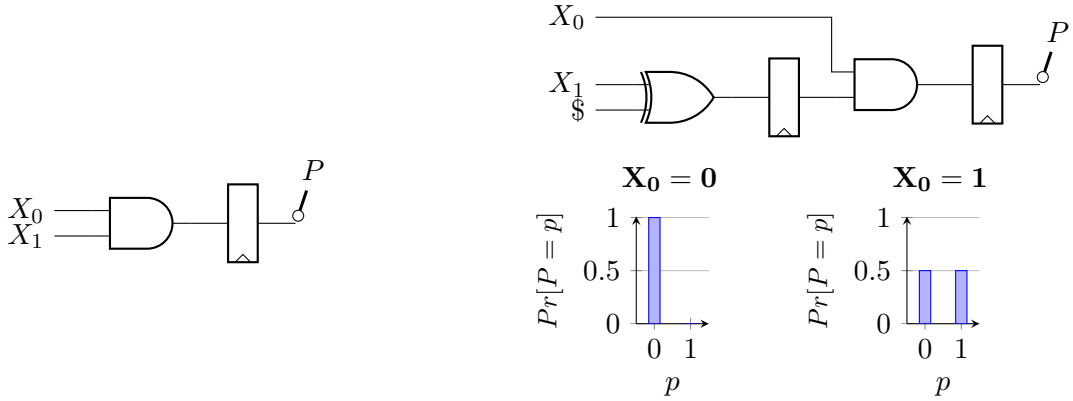
---

**Input:**  $\mathcal{S} = \{X_0 \in \mathbb{F}_2\}$

**Output:**  $P \in \mathbb{F}_2$ : Simulated probe observation

```
if  $X_0 = 0$  then
    return  $P \leftarrow 0$ 
else
    return  $P \xleftarrow{\$} \mathbb{F}_2$ 
end if
```

---



(a) Both shares necessary to perfectly simulate the distribution observed by  $P$       (b) Only a single share necessary to perfectly simulate the distribution observed by  $P$

Figure 2.6: Perfect simulatability example: Probe placement on exemplary minimal circuits

into a wire if the signal on that wire is required to perfectly simulate the probe’s view, i.e., mimicking the observations made by the probes over any number of circuit executions.

Intuitively, the concept describes how leakage traverses backward through the circuit, starting from the location where a probe is placed. This enables formal arguing about which input shares a set of probes can get information. Fundamentally, for the resulting overall circuit, it comes down to ensuring that any set of probes on the circuit only propagates into a part of all shares and not all of them. If the input masking fulfills Definition 2.4.2, this will not reveal any information about the secret itself. Note that restricting probe propagation to subsets of all shares is not equivalent to guaranteeing  $d$ -probing security but may impose stronger restrictions onto a masked circuit. This means it is indeed a sufficient requirement, but not a necessary one.

A simple example for a probe whose observations are independent of the secret but which propagates into all shares of a secret is given in Table 2.1. If we assume that  $X_0, X_1, Y_0$  are the result of a uniform masking, we have that

$$Pr[P = 1|X = x] = Pr[P = 1] = 1/4 \text{ and } Pr[P = 0|X = x] = Pr[P = 0] = 3/4, \quad \forall x \in \{0, 1\},$$

and hence independence between the secret  $X$  and probe  $P$ . Nonetheless, the probe cannot be perfectly simulated according to Definition 2.6.2 without knowledge of both  $X_0$  and  $X_1$ .

Ultimately, this conservative approach abstracts from the information-theoretic argument of independence between random variables and makes constructing and arguing about security much easier, but may introduce unnecessary overhead into a circuit. Over time, various composability notions have been proposed, all of which aim to restrict probe propagation within gadgets sufficiently so that  $d$ -probing security is guaranteed when they are interconnected and form a larger circuit. We introduce them below.

**Non-Interference.** The fundamental idea of composability notions is the restriction of probe propagation within gadget boundaries. Here, it is ensured that every possible set of probes placed onto a gadget only propagates into a limited set of input shares. All proposed composability notions aim to sufficiently restrict such sets in order to guarantee secure composition.

$X$	$X_0$	$X_1$	$Y_0$	$P = (X_0 \oplus Y_0)X_1$
0	0	0	0	0
0	0	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0

Table 2.1: Example probe which is independent of the secret  $X$  but propagates into  $X_0$  and  $X_1$ .

The idea is to limit gadget-individual probe propagation in a way that ultimately restricts propagations of global probes placed on the interconnected circuit in such a way that only information of a proper subset of all shares of each initial secret inputs is leaked, hence ensuring (robust)  $d$ -probing security. The NI notion, introduced by Barthe et al in [BBD<sup>+</sup>15], ensures that any set of  $t \leq d$  probes only propagates into at most  $t$  shares per input, which is captured by Definition 2.6.3.

**Definition 2.6.3** ( $d$ -Non-Interference (NI) [BBD<sup>+</sup>15]). A masked circuit  $\tilde{\mathcal{C}}$  provides  $d$ -Non-Interference if and only if for any probe set  $\mathcal{P}$  of  $t \leq d$  probes, there exists a set  $\mathcal{S}$  of input shares with  $|\mathcal{S}^j| \leq t$  for all input indices  $j$  such that  $\mathcal{P}$  can be perfectly simulated by  $\mathcal{S}$ .

**Strong Non-Interference.** As the NI notion was not sufficient to guarantee secure composition of gadgets, probe propagation was further restricted by a novel extension, the Strong Non-Interference (SNI) notion [BBD<sup>+</sup>16]. In this context, probes placed onto output wires do not increase the number of shares available for simulation. Intuitively, this stops the propagation of probes placed onto composed circuits at gadgets fulfilling the SNI property. This is formalized in Definition 2.6.4. One drawback of SNI is that the trivial implementation, i.e., share-wise application, of linear functions is not SNI, and special care must be taken to omit critical probe propagation by inserting SNI refresh gadgets into the composed circuit.

**Definition 2.6.4** ( $d$ -Strong Non-Interference (SNI) [BBD<sup>+</sup>16]). A masked circuit  $\tilde{\mathcal{C}}$  provides  $d$ -Strong Non-Interference if and only if for any probe set  $\mathcal{P}$  containing  $t = t_1 + t_2 \leq d$  probes, where  $t_1$  probes are placed on internal wires and  $t_2$  on output wires, there exists a simulation set  $\mathcal{S}$  of input shares with  $|\mathcal{S}^j| \leq t_1$  for all input indices  $j$  such that  $\mathcal{P}$  can be perfectly simulated by  $\mathcal{S}$ .

**Probe-Isolating Non-Interference.** The concept of Probe Isolating Non-Interference (PINI) was introduced by Cassiers et al. in [CS20] with the goal of reducing the overall overhead of composed circuits compared to the SNI context. According to Definition 2.6.5, very similar to Domain-Oriented Masking (DOM) [GMK16], share domains are introduced, and internal probes placed on a gadget are restricted to propagating into an arbitrary single share domain. In contrast, output probes are only allowed to propagate within the same share domain as the



output share on which the probe was placed. Compared to NI/SNI, this novel composability notion has the significant advantage of allowing the trivial realization of linear functions. More precisely, the share-wise application of linear functions is directly consistent with the PINI notion. Thus, in contrast to NI/SNI, they do not introduce any additional overhead into the design, except for signal expansion and function duplication. No additional registers or fresh random bits are required. Furthermore, the composition of gadgets that satisfy the PINI notion is straightforward. As long as wires of gadgets are connected with respect to equal share domains, the composed circuit will be secure in the  $d$ -probing adversary model, which greatly simplifies the gadget-based construction of masked circuits.

**Definition 2.6.5** ( $d$ -Probe Isolating Non-Interference (PINI)). Let  $\mathcal{P}_I$  be the set of internal probes with  $|\mathcal{P}_I| = t_1$ . Furthermore, let  $\mathcal{I}_O$  be the index set assigned to the output wires probed by  $\mathcal{P}_O$  with  $|\mathcal{I}_O| = t_2$ .

A masked circuit  $\tilde{C}$  provides  $d$ -Probe Isolating Non-Interference if and only if for every  $\mathcal{P} = \mathcal{P}_I \cup \mathcal{P}_O$  with  $t_1 + t_2 \leq d$ , there exists a set  $\mathcal{I}_I$  of circuit indices with  $|\mathcal{I}_I| \leq t_1$  such that  $\mathcal{P}$  can be perfectly simulated by input shares with indices drawn only from  $\mathcal{I}_I \cup \mathcal{I}_O$ .

### 2.6.5 Gadget Realizations

Over time, various composable hardware gadgets have been proposed that satisfy different security notions and aim to optimize for different overhead parameters. An overview of common gadget realizations in the context of the  $d$ -probing model is given in Table 2.2.

**HPC1.** HPC1 was introduced in [CGLS21] and is essentially an extension of a simple DOM multiplier [GMK16]. To achieve composability under the notion of PINI, a SNI refresh was inserted for one of the inputs, preceding the actual DOM multiplier. As a consequence, compared to the original DOM multiplier, additional fresh randomness is introduced, the amount of which is determined by the security order  $d$ .

**HPC2.** Also in [CGLS21], HPC2 was introduced as a PINI-composable gadget that essentially lowers the randomness requirements of HPC1 by cleverly combining shares to compute cross-domain terms without the need for a dedicated SNI refresh of any of the inputs. It requires a total of two register stages. An additional advantage of HPC2 is the asymmetry in input-output latency. One of the gadget's inputs is only needed in the combinatorial logic between the first and second register stages, while the other is needed before the first register stage. It has been shown in [CGLS21] that this asymmetry can be exploited to achieve a favorable total latency of composed overarching designs, meaning that the total latency can be less than twice the maximum number of AND gates in any path of a circuit.

**HPC3.** In [KM22b], we introduce a low-latency hardware private circuit. Similar to HPC1 and HPC2, it implements an AND functionality and can be instantiated for arbitrary security orders. HPC3 has a favorable trade-off with respect to latency, since it requires only a single register stage, but uses twice as many random bits as HPC2.

Gadget	Notion	$d$	Function	#Random Bits	Latency
HPC1 [CGLS21]	PINI	arbitrary	AND	$d(d+1)/2 + r[d]^*$	2
HPC2 [CGLS21]	PINI	arbitrary	AND	$\frac{d(d+1)}{2}$	2
<b>HPC3</b> [KM22b]	PINI	arbitrary	AND	$d(d+1)$	1
<b>GHPC</b> [KSM22]	PINI	1	$\mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$	$m$	2
<b>GHPC<sub>LL</sub></b> [KSM22]	PINI	1	$\mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$	$m \cdot 2^n$	1
ISW-AND [FGP <sup>+</sup> 18]	SNI	arbitrary	AND	$d(d+1)/2$	2
CMS <sub>LL</sub> <sup>†</sup> [MPZ22]	SNI	arbitrary	AND	$2(d+1)^2$	1

\*  $r[d] = [1, 2, 4, 5, 7, 9, 11, 12, 15, 17]$  for  $d \leq 10$

† hand-optimized for low security orders; number of randomness = [2,6,12] for  $d = [1,2,3]$  respectively

Table 2.2: Overview of different nonlinear hardware gadgets; constructions that are part of this thesis are marked in **bold**.

**GHPC.** In [KSM22], we introduce a novel methodology for creating gadgets that are securely composable for the first security order under the notion of PINI. With this technique, a circuit with arbitrary functionality can be transformed into a masked gadget solely based on its functional description, e.g., its Algebraic Normal Form (ANF). GHPC comes in two variants: (i) the standard variant, which requires two clock cycles to perform any functionality and uses only a single bit of randomness per coordinate function of the vectorial Boolean function, and (ii) a low-latency variant – referred to as **GHPC<sub>LL</sub>** – which requires only a single clock cycle regardless of the realized functionality, but uses  $2^n$  random bits per coordinate function, where  $n$  is the number of unshared inputs.

**ISW-AND.** In order to compose probing-secure circuits, SNI-compliant ISW-AND gadgets can be used, which were introduced in a glitch-robust variant in [FGP<sup>+</sup>18]. In contrast to the PINI notion, in the context of SNI, refresh gadgets must be carefully inserted into the composed circuit to prevent security-degrading probe propagation. As shown in [BBD<sup>+</sup>16, CGLS21], trivial, i.e., share-wise, implementations of linear operations are not SNI. Depending on the circuit structure, this may demand for additional robust-SNI refresh gadgets to be inserted into the circuit which further increases the introduced overhead. As the original composition approach by Barthe et al. [BBD<sup>+</sup>16] was too conservative considering the insertion of refresh gadgets, Belaïd et al. introduced a new composition strategy in [BGR18], drastically reducing the number of required refresh gadgets.

**CMS<sub>LL</sub>.** In [MPZ22], an adaption of Consolidating Masking Schemes (CMS) [RBN<sup>+</sup>15] was introduced to construct an SNI-composable hardware gadget that realizes a masked variant of an AND gate and only needs a single register stage in total. Similar to ISW-AND, a careful combination of CMS<sub>LL</sub> and SNI refresh gadgets allows the systematic construction of masked circuits secure in the (robust)  $d$ -probing model.

### 2.6.6 Discussions on the Costs of Automated Masking

As discussed above, automated masking adds significant overhead to a design. In the following, we will briefly discuss the general costs of automated masking.

As cost metrics, we consider the on-chip area footprint of implementations, the latency in clock cycles and the randomness requirements as the number of coin flips. Note that the randomness requirements will eventually translate into area overhead, as fresh random bits originate from some form of randomness source realized as a dedicated hardware circuit on the chip. In practice, the randomness supply is commonly realized by a Pseudo-Random Number Generator (PRNG) whose area grows with the output throughput per clock cycle. Consequently, the randomness requirement indirectly translates into area overhead, but since there is no well-defined cost function for the throughput of a randomness source, quantifying the cost in a general way is currently impossible, and elaborations are based on specific case studies.

In general, for two-input nonlinear gadgets (with input  $X$  and  $Y$ ), e.g.,  $\text{HPC}\{1, 2, 3\}$ ,  $\text{ISW-AND}$  and  $\text{CMS}_{\text{LL}}$ , the area (and randomness) costs of each gadget scale within  $\mathcal{O}((d+1)^2)$ , as both are mainly determined by deriving the cross terms  $X_i \cdot Y_j$  with  $i \neq j$  (the multiplication of shares from different domains), while restricting the probe propagation in accordance with their respective composability notions. In this case, the area of a composed circuit is mainly driven by the number of nonlinear gates in the unprotected circuit, while the total latency is determined by the maximum number of nonlinear gates in any input-output path of the circuit.

$\text{GHPC}$  and  $\text{GHPC}_{\text{LL}}$  pose an exception as they enable the construction of low-randomness and low-latency construction for the first security order. Here, the cost is driven by the number of inputs into the circuit. Note that the area consumption of both increases exponentially with the number of unshared inputs to the circuit. This is the price we have to pay for guaranteeing a fixed latency independent of the complexity of the Boolean function.

Generally, there exists no one-fits-all solution, i.e., one gadget that is superior with respect to all overhead metrics. It is always a trade-off between area, latency and randomness and the gadgets have to be selected with care given a specific use case.

## 2.7 Formally Verifying SCA Resistance and Secure Composability

Attesting SCA resilience early on in the process of IC design is highly beneficial, as it allows to check security before the actual production of the chip.

### 2.7.1 Pre-Silicon Verification Methodologies for Hardware

Pre-silicon verification of the SCA resilience of cryptographic hardware implementations has significant advantages over experimental leakage assessments as described in Section 2.3.4. It eliminates the costly iteration from masking, through production, to experimental verification, and allows for rapid and cost-effective pre-production security verification. Several methods and tools have been proposed for this purpose, which we summarize below. There exist many more tools that are not checking for security in a formal adversary model like tools based on RTL-based simulation [ZSHS12, HS17, HPN<sup>+</sup>19, PPFT22, LRB22] and low-level power simulation [YTK<sup>+</sup>21, KYL<sup>+</sup>22, KLS22, MZG<sup>+</sup>22]. Moreover, there are tools settled in the random probing model [BCP<sup>+</sup>20] or tools that check the composition-based security, i.e., the correct interconnection of gadgets [BGR18, BDM<sup>+</sup>20, CGLS21]. Here, we focus on gate-level

verification in the robust  $d$ -probing model and refer the interested reader to [FSG22] for an excellent survey.

**Rebecca [BGI<sup>+</sup>18].** Rebecca is a verification tool that is based on the netlist description of a masked circuit. It checks for security in the  $d$ -probing model while it models glitches differently than the robust probing model introduced in [FGP<sup>+</sup>18]. Its methodology is based on combining Fourier transformation and elaborating correlation through a Satisfiability (SAT)-solver. This technique can lead to false negative results, i.e., there are designs that are claimed to be insecure by Rebecca, but which are actually in conformity with the  $d$ -probing model.

**maskVerif [BBD<sup>+</sup>15].** maskVerif performs formal verification within the  $d$ -probing model. For this, it utilizes a symbolic representation of leakage which is defined by a given syntax and semantics. Internally, it then uses a semantic-preserving transformation of leakage sets before it concludes secret dependencies. As well as Rebecca, the underlying approach can lead to secure designs being falsely claimed as insecure. The approach was later picked up in different works [BBD<sup>+</sup>16, Cor18, BBC<sup>+</sup>19, BGG<sup>+</sup>21, BK21] with the aim to increase verification efficiency, extend the scope of functionality or omit false negative results.

**SILVER [KSM20].** SILVER – which is part of this thesis – was introduced as the first tool to support comprehensive evaluation in the (robust)  $d$ -probing model in the presence of glitches. It not only supports the verification of a circuit’s conformance to  $d$ -probing security, but also allows the verification of all common composability notions introduced in Section 2.6.4. For the actual verification, it uses BDDs to compute and compare distributions.

**IronMask [BMRT22].** IronMask can formally verify masked circuits in the (robust)  $d$ -probing model and the random probing model. For each possible probe placement, IronMask identifies a necessary and sufficient set of inputs for a perfect leakage simulation. In addition to verifying the security of  $d$ -probing and random probing itself, it is capable of verifying compliance with all common notions of composability. It is thus a complete tool for the formal verification of hardware circuits.

**PROLEAD [MM22].** PROLEAD is a simulation-based tool that verifies a circuit’s compliance with the  $(1, 1, 0)$ -robust  $d$ -probing model, i.e., it considers glitches and transitions at register stages, but does not perform an exhaustive verification for each and every input. Instead, it simulates the circuit for different (shared) inputs and checks for dependencies between the observations of probes placed on wires derived with respect to the  $(1, 1, 0)$ -robust  $d$ -probing model, and the secret. The dependency itself is elaborated by a  $G$ -test [Hoe12].

### 2.7.2 Binary Decision Diagram

Binary Decision Diagrams (BDDs) are data structures utilized for presenting Boolean functions. Common applications can be found in EDA where they are used for logic synthesis and digital circuit analysis.

**Shannon's Expansion.** *Shannon's Expansion* [Sha49b] – also called *Shannon Decomposition* – offers a well-defined format to represent any Boolean function. A definition is given in Definition 2.7.1.

**Definition 2.7.1** (Shannon's Expansion [Sha49b]). Let  $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$  be a Boolean function. Then Shannon's decomposition of  $f(X_0, X_1, \dots, X_{n-1})$  with respect to  $X_i$  is given as

$$\begin{aligned} f(X_0, X_1, \dots, X_{n-1}) &= X_i \cdot f_{X_i} \vee \overline{X_i} \cdot f_{\overline{X_i}}, \quad \text{with} \\ f_{X_i} &\equiv f(X_0, \dots, X_{i-1}, 1, X_{i+1}, \dots, X_{n-1}), \\ f_{\overline{X_i}} &\equiv f(X_0, \dots, X_{i-1}, 0, X_{i+1}, \dots, X_{n-1}). \end{aligned} \tag{2.2}$$

We refer to  $f_{X_i}$  and  $f_{\overline{X_i}}$  as Shannon co-factors of the expansion. Straightforwardly, we can derive an alternative, but still valid, expansion by replacing the logic OR by an XOR because both terms  $X_i \cdot f_{X_i}$  and  $\overline{X_i} \cdot f_{\overline{X_i}}$  in Equation 2.2 cannot be true at the same time. It is possible to further expand a Boolean formula by recursively applying the decomposition on the derived Shannon co-factors.

As an example, we can consider  $f = X_0X_1 \vee X_0X_2 \vee X_1$ . Shannon's expansion with respect to  $X_0$  would lead to:

$$f(X_0, X_1, X_2) = X_0 \cdot (X_1 \vee X_2) \vee \overline{X_0} \cdot (X_1)$$

Now, if we continue the expansion recursively, this would lead to

$$\begin{aligned} f(X_0, X_1, X_2) &= X_0 \cdot (X_1 \vee X_2) \vee \overline{X_0} \cdot (X_1) \\ &= X_0X_1 \cdot (X_2 \vee 1) \vee X_0\overline{X_1} \cdot (X_2) \vee \overline{X_0}X_1 \cdot (1) \vee \overline{X_0}\overline{X_1} \cdot (0) \\ &= X_0X_1\overline{X_2} \vee X_0\overline{X_1}X_2 \vee \overline{X_0}X_1. \end{aligned}$$

**Binary Decision Diagram.** Binary Decision Diagrams (BDDs) [Jr.78, Bry86] are commonly applied data structures in the context of computer science and discrete mathematics and are based on the fact that each Boolean function  $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$  can be represented as a rooted, directed, acyclic binary graph consisting of decision nodes. Each node – except the leaves – is labeled by an input variable and has exactly two children. One outgoing edge represents the variable taking value 0 while the other edge represents value 1. Hence, each node can be seen as the Shannon decomposition of the subgraph of which the node is the root. The leaves of the BDD represent the values the function can take, i.e., it is labeled either 0 or 1. Eventually, each path in the graph represents a function evaluation with respect to a particular input.

Furthermore, a BDD can be *reduced* or *ordered*, which is defined in Definition 2.7.2 and Definition 2.7.3, respectively. It can also be both reduced and ordered as defined in Definition 2.7.4.

**Definition 2.7.2** (Ordered Binary Decision Diagram (OBDD) [Bry86]). An Ordered Binary Decision Diagram (OBDD) is an BDD where on each path, the input variables are traversed in the same order.

**Definition 2.7.3** (Reduced Binary Decision Diagram (RBDD) [Bry86]). A Reduced Binary Decision Diagram (RBDD) is derived from a BDD by:

- (1) Merging isomorphic subgraphs

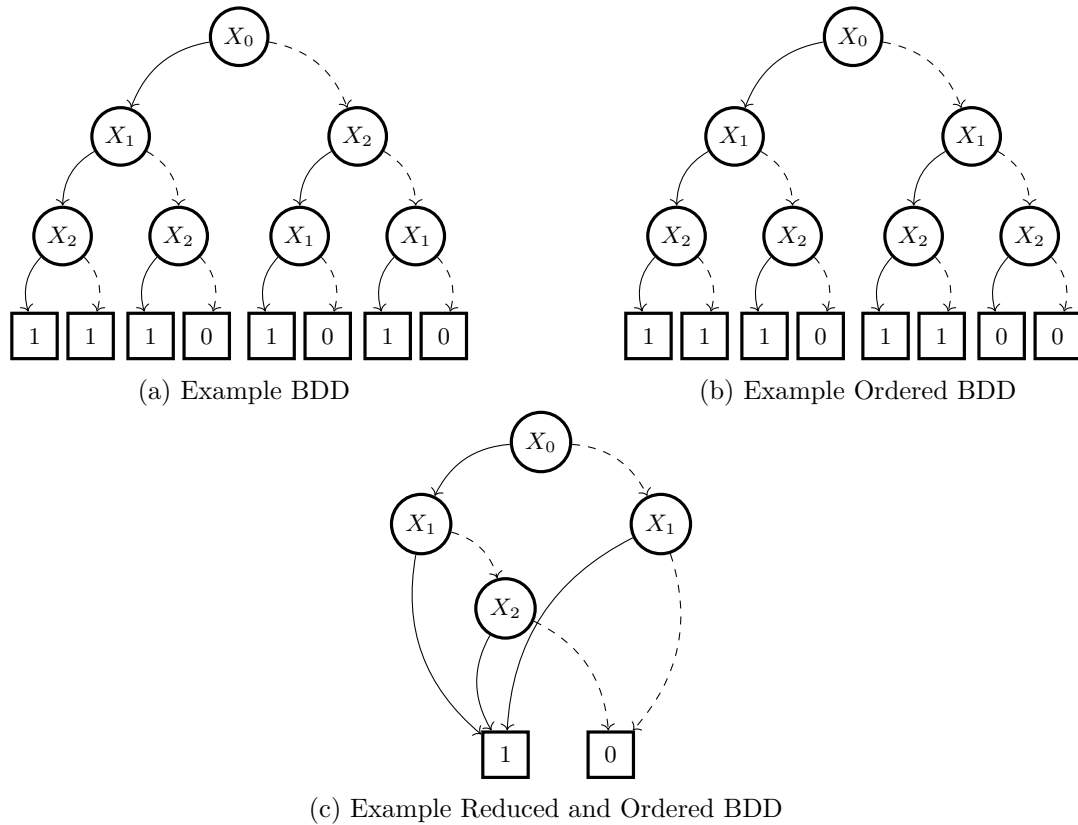


Figure 2.7: From BDD to Reduced and Ordered BDD of  $f(X_0, X_1, X_2) = X_0X_1 \vee X_0X_2 \vee X_1$

(2) Erasing any node whose two children are isomorphic

**Definition 2.7.4** (Reduced Ordered Binary Decision Diagram (ROBDD) [Bry86]). A Reduced Ordered Binary Decision Diagram (ROBDD) is a BDD which is a RBDD *and* an OBDD.

Considering our example function  $f(X_0, X_1, X_2) = X_0X_1 \vee X_0X_2 \vee X_1$ , a corresponding unordered and unreduced BDD can be seen in Figure 2.7a, while an OBDD is depicted in Figure 2.7b. An ROBDD of  $f$  can be seen in Figure 2.7c.

**Part II**

**Publications**





# Chapter 3

## Persistent Threat of Side-Channel Analysis Attacks

*In this chapter, we present a peer-reviewed publication as part of this thesis which underlines the persistent threat of SCA attacks. In particular, we present our paper published in the IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHEES).*

### Contents of this Chapter

---

<b>3.1 Let's Take it Offline: Boosting Brute-Force Attacks on iPhone's User Authentication through SCA</b> . . . . .	<b>49</b>
--	-----------

---

### 3.1 Let's Take it Offline: Boosting Brute-Force Attacks on iPhone's User Authentication through SCA

#### Publication Data

Oleksiy Lisovets, David Knichel, Thorben Moos, and Amir Moradi. Let's Take it Offline: Boosting Brute-Force Attacks on iPhone's User Authentication through SCA. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):496–519, 2021

*This work is reproduced here with permission. This work is licensed under a Creative Commons Attribution 4.0 International License. Copyright is held by the authors. The author of this thesis is also an author of this research paper.*

**Content.** In this work, we extract hardware-fused key material from an iPhone 4 by combining publicly available software exploits and SCA attacks. For this, we first gain on-device code execution by means of a BootROM exploit, which allows us to obtain oracle access to the AES core of the iPhone. Afterwards, leveraging this oracle access, we run an SCA attack. In particular, we perform successful CPAs on the GID and UID key of the device. For this, we compare the efficiency of our attack when collecting EM traces to the performance when measuring the power consumption of the device. We show that having the UID key in hand allows to perform a passcode search offline. As a result, we are able to achieve a massive speed-up compared to the on-device search by running the passcode search in parallel on multiple GPUs.

**Contribution.** The author was involved in conducting the key recovery attack. Furthermore, the author significantly contributed to the writing and the presentation of the results. The author would like to thank all three co-authors for their substantial contributions and particularly like to thank Oleksiy Lisovets, whose hard work during his master thesis – which was supervised by the author of this thesis – led to this publication.

# Let's Take it Offline: Boosting Brute-Force Attacks on iPhone's User Authentication through SCA

Oleksiy Lisovets<sup></sup>, David Knichel<sup></sup>, Thorben Moos<sup></sup> and Amir Moradi<sup></sup>

Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany,  
 [{firstname.lastname}@rub.de](mailto:{firstname.lastname}@rub.de)

**Abstract.** In recent years, smartphones have become an increasingly important storage facility for personal sensitive data ranging from photos and credentials up to financial and medical records like credit cards and person's diseases. Trivially, it is critical to secure this information and only provide access to the genuine and authenticated user. Smartphone vendors have already taken exceptional care to protect user data by the means of various software and hardware security features like code signing, authenticated boot chain, dedicated co-processor and integrated cryptographic engines with hardware fused keys. Despite these obstacles, adversaries have successfully broken through various software protections in the past, leaving only the hardware as the last standing barrier between the attacker and user data. In this work, we build upon existing software vulnerabilities and break through the final barrier by performing the first publicly reported physical Side-Channel Analysis (SCA) attack on an iPhone in order to extract the hardware-fused device-specific User Identifier (UID) key. This key – once at hand – allows the adversary to perform an offline brute-force attack on the user passcode employing an optimized and scalable implementation of the Key Derivation Function (KDF) on a Graphics Processing Unit (GPU) cluster. Once the passcode is revealed, the adversary has full access to all user data stored on the device and possibly in the cloud. As the software exploit enables acquisition and processing of hundreds of millions of traces, this work further shows that an attacker being able to query arbitrary many chosen-data encryption/decryption requests is a realistic model, even for compact systems with advanced software protections, and emphasizes the need for assessing resilience against SCA for a very high number of traces.

**Keywords:** iPhone · SCA · Passcode Recovery

## 1 Introduction

Over the last decade, smartphones entirely reshaped the way we communicate while drastically increasing the amount of user-related data collected and stored on device or uploaded into the cloud. With the advancement of the Internet of Things and increased interconnection of everyday-life devices, this trend is reinforced even further. Smartphones became a replacement for various smart cards including credit cards; they process health-related data, and serve as an access key in various applications. As a result, compromising the security and data protection mechanisms of smartphones dramatically impacts user privacy. Here, the user authentication plays a key role as its bypass would lead to full control over the device. While biometric-based authentication methods like fingerprint matching or face recognition were introduced over the years, using a (numeric) passcode is still commonly the fall-back option if these methods fail. Next to Samsung and Huawei

devices, Apple's iPhone is the most sold smartphone in the world showing a market share of 14% in the first quarter of 2020 [Goa20]. With the introduction of Touch ID in iPhone 5s and later with Face ID for the iPhone X, Apple has started integrating biometric-based user authentication methods in their devices while setting a passcode still remains as a prerequisite to enable Touch ID or Face ID. Whereas a 4-digit numeric passcode was the default option for older system versions, its length has been extended to contain 6 numeric digits with the launch of iOS 9. Nevertheless, iOS always offers an option for manually setting an arbitrary-length ( $\geq 4$ ) numeric/alphanumeric passcode. We should highlight that it is always possible – and in case of a cold boot even required – to unlock the iPhone using the passcode instead of the configured biometric authentication methods. Therefore, the passcode is still the default fall-back solution for user authentication, enabling an adversary to alternatively target this method in order to maliciously gain access to the device's system and user data.

When the user enters the passcode, it is entangled with a device-specific key called User Identifier (UID). The UID is an encryption key unique for each device and is fused into hardware during manufacturing. The result of running the aforementioned function is then used to decrypt/encrypt user data stored on the device. The entanglement between passcode and the UID key restricts brute-force attempts to be performed on the (same) device. In addition, a single execution of the aforesaid cryptographic function is designed to take about 80 milliseconds, resulting in an expected time of over 22 hours for an on-device brute force search of a 6-digit numeric passcode.

Current methods, like the one presented by Markert et al. in [MBG<sup>+</sup>20], try to optimize the on-device search of a user's passcode through an ordered list of common PINs, as PIN verification is tied to a specific phone. By utilizing Side-Channel Analysis (SCA) attacks, we show that the coupling of the brute-force search to a specific iPhone can be levered out completely, enabling a significantly faster passcode recovery.

Since the first introduction of SCA attacks as a threat to cryptographic implementations [Koc96, KJJ99], researchers and practitioners have reported successful key-recovery attempts mainly on their-own-designed devices. This picture has changed when they have been shown highly effective by targeting real world devices, even in a blackbox scenario, where no details about the implementation are known to the adversary. For example, in [EKM<sup>+</sup>08, KKMP09] SCA attacks were used to completely break the remote keyless entry system based on KeeLoq technology employed by several car manufacturers at that time. In [ORP13], by means of SCA, the authors reported the recovery of the Yubike2's secret key – a One-Time Passwords (OTPs) token used for two factor authentication – enabling the malicious generation of valid OTPs, even after returning the token to the owner. In 2011, the result of a power analysis attack on the contactless smartcard DESFire MF3ICD40 was reported [OP11], resulting in a complete recovery of its 112-bit key. As this smartcard was employed in several large payment and public transport systems around the world at that time (e.g., Czech railway, Australian myki card, Clippercard in San Francisco) it evidently emphasizes the relevance of SCA attacks in real-world scenarios. Further, in [SRH16], Saab et al. show two ways of recovering AES keys in the context of Intel's AES-NI cryptographic instruction set extension by placing a magnetic field probe in close proximity of two capacitors on a motherboard hosting an Intel Core i7 Ivy Bridge microprocessor.

An example for a successful SCA attack performed on a smartphone is presented in [BFMT16], where Belgarric et al. successfully recovered the key used in the implementation of the ECDSA signature scheme in Android's standard cryptographic library. In their work, they leverage the electromagnetic emanation of the CPU to distinguish between different elliptic curve operations in the context of Weierstrass and Koblitz Curves. Another EM-based SCA attack on a smartphone is reported in [VMC19], where the authors successfully extract the secret AES key of the CPU's hardware coprocessor. The work

especially focuses on issues arising when performing such an attack on a modern system and involves desoldering the DRAM placed on top of the main SoC. However, the manufacturer and model of the device under test are not disclosed.

Academic publications dealing particularly with physical attacks on iPhones are hard to find. To the best of our knowledge only a 2016 work by Sergei Skorobogatov [Sko16] describes a real-world implementation attack on an iPhone. In more detail, the author describes how to perform a real-world mirroring attack on an iPhone 5c, enabling to bypass the limit of passcode retry attempts by restoring a previous state of the NAND flash memory. Beyond that, only informal reports of successful hardware exploitation of iPhone devices exist. For instance, an article from *The Intercept*, published in 2015 in course of the Snowden Leaks, mentions that already in 2011, EM-based SCA attacks were performed by the CIA in order to extract the Group Identifier (GID) key from iPhone 4 devices in order to decrypt and analyze the boot firmware for vulnerabilities [SB15]. More details can be found in [int15].

In previous works by Sogeti [sog11b] and Elcomsoft [elc11], the software implementation of iPhone data protection has been reverse engineered. Furthermore, they utilized a BootROM vulnerability to perform on-device passcode cracking by booting a custom Secure Shell (SSH) ramdisk with a patched kernel, allowing them to instruct the Advanced Encryption Standard (AES) engine to use the UID key from userspace.

In this work, we utilize a BootROM vulnerability to deploy a custom payload in bootloader context, which allows us to communicate with the AES engine and perform a successful SCA attack on an iPhone 4 recovering the complete 256-bit UID key fused into the hardware design of an individual device’s processor. Despite the fact that no detailed information about the attack procedure is included in the leaked documents [SB15, int15], it is likely that the attacks supposedly performed by the CIA to extract the GID key from an Apple A4 processor, are very similar to our attack presented in this work. The hardware AES engine can be configured to use either of the two keys, GID or UID, so that the attack procedure for extracting them is identical. In our attack, having the UID key in hand, the authentication process becomes decoupled from the device, enabling a significantly faster and scalable brute-force search of the passcode utilizing highly parallelized computation on Graphics Processing Units (GPUs). Since we show that existing software vulnerabilities enable the collection and processing of a high number of measurement traces, our work underlines the practical relevance of assessing the resilience of a device against SCA, even for several hundreds of millions of traces, which is often questioned by the research community.

For the collection of the SCA measurements, the device evidently needs to be physically accessed by the adversary. Afterwards, however, the adversary does not have to be in the possession of the iPhone anymore, as the storage can be dumped and the passcode search is performed completely offline. In order to give an overview on the performance of our attack, we refer to the required time for a trivial on-device brute-force search of an 8-digit numeric passcode which is reduced from 92 days to less than 3 hours by performing the search on 2 NVIDIA RTX 2080 TI GPUs [NVI18].

The rest of the paper is structured as follows. First, we give an overview of the attack scenario and our approach in Section 2, before all necessary background is clarified in Section 3. In Section 4, we describe how we recovered the UID key by performing SCA attacks on an iPhone 4. Once recovered, we use this key to perform a parallel brute-force search of the passcode, as detailed in Section 5. A discussion on the applicability of this attack to newer iPhone series is given in Section 6. Finally, we conclude the paper in Section 7.

## 2 Attack at a Glance

The attack procedure described in the following is based on the assumption that code execution by means of a software exploit, i.e., a BootROM exploit, is achieved. The adversary then has oracle access to the AES engine realized as a dedicated hardware co-processor on the CPU and can query arbitrary data to be encrypted in a chosen-plaintext SCA attack. For our device under test, the necessary BootROM exploit is publicly available.

### 2.1 Attack Scenario

First, the adversary maliciously gains physical access to the victim's device. Without any persistent modification of the system, she then boots a custom ramdisk providing SSH access and downloads the System Keybag from device (further described in Section 3.2). Furthermore, she performs an SCA attack in order to extract the device-specific UID key. For user authentication, the UID key is entangled with the user's passcode in order to derive the so-called passcode key which is in turn used to unwrap further keys from the System Keybag. This can then be used to unlock individual files stored encrypted on the filesystem. Using existing vulnerabilities, we show that the attacker can easily extract this file from the device – again without any permanent changes on the system – which, together with the UID key, enables the recovery of the user's passcode by means of an offline brute-force search. After successful recovery of the passcode, the system can be normally booted and the adversary can simply log into the device, allowing complete access to all data, e.g. by running an iTunes backup. As the necessary modifications of the mainboard only consist of removing the metal shielding of the CPU (which can be simply unclipped and clipped back), they can easily be reversed and the phone can still be operated normally after the attack, leaving the victim with no direct chance to recognize any malicious activity. The power analysis attacks which we performed in addition to the EM-based attacks, require the disassembly of inductors and capacitors from the board, but even in that case it is possible to reverse the modifications, although greater care and effort is required. Admittedly, the attack procedure requires not only physical access and minor modifications of the hardware, but also a considerable amount of time to acquire the necessary amount of measurements (in our case, it took three weeks in total to recover the UID). Nonetheless, we can think of highly relevant attack scenarios where an attacker is able to invest the time for this attack and where it is worthwhile to do so. Apart from finding or stealing a device and extracting all the sensitive user data, an attacker could also run a malicious phone shop where she extracts the UID keys in advance before handing out the phone to the customer and might sell them to malicious third parties. In fact the break-even point, where the time spent to perform the physical attack is less than the time saved when doing offline instead of on-device brute-force search is reached for passcodes of 8 digits and more.

In this work we present the aforementioned attack on an iPhone 4; however the same setup (with minor changes) can be used to collect SCA measurements and extract the keys from an iPhone 4s and iPhone 5/5c. We further believe that similar research can be done on devices up to the iPhone X/iPhone 8. In fact, due to a couple of publicly-known vulnerabilities [ax, Xu] of such newer devices (explained in detail in Section 6), it might be possible to obtain oracle access to the relevant AES engine and collect the SCA measurements necessary for an attack. However, since we have not practically attempted such experiments yet, we cannot make any claims about the success/difficulty of the attacks on newer devices, especially considering that newer iPhones reportedly contain DPA countermeasures.

## 2.2 High-Level Description

At first, we execute the SHA-1 Image Segment Overflow (SHAtter) exploit to disable signature checks in the initial Secure Boot process, enabling the execution of custom codes. Afterwards, we deploy a payload integrated into the second-stage bootloader, enabling us to query an arbitrary amount of chosen data to the AES hardware engine for encryption and decryption, and to re-configure one of the General Purpose Input/Output (GPIO)-pins (e.g. volume-down button) as output for triggering the SCA measurements. By performing a Correlation Power Analysis (CPA), we exploit correlation between the recorded SCA traces and secret intermediate values of the AES encryption, allowing us to recover the complete 256-bit UID key. We show that being in possession of this key allows an offline recovery of the user passcode by running a parallelized brute-force search on several GPUs. Having recovered the passcode in turn enables a decryption of all user files previously stored on the device and potentially (if the user is signed in to iCloud on the device) data stored on the user's iCloud.

## 3 Background

### 3.1 Secure Boot

As described in [App12], the Apple iPhone 4 implements a secure boot mechanism to establish a boot-chain-of-trust which starts at the BootROM and works its way up to the operating system kernel and even further to application level. Here we focus only on the first boot components relevant to our attack.

On power up, the BootROM, which is an immutable piece of software fused into the System on Chip (SoC), is executed. This is the root of the trust chain. The BootROM functionality is kept minimal as it is the most trusted code and a vulnerability in the BootROM cannot be fixed with a software update. Its task is to load, validate and execute the first stage bootloader either from Non-Volatile Random Access Memory (NVRAM) or over Universal Serial Bus (USB). The latter is used in cases where the former fails or the user enters the Device Firmware Upgrade (DFU) mode using a special button combination.

The first stage bootloader runs in the Static Random Access Memory (SRAM) with its main responsibility being the initialization of low-level hardware components such as Dynamic Random Access Memory (DRAM) and to load, validate and execute the next stage of the bootloader. As it is executed in DRAM, the second stage is provided with much more memory capacities. It is responsible for initializing higher-level hardware components (such as the screen) and further for loading – amongst other parts – the boot logo, the devicetree and a ramdisk (in case of a recovery/update boot). Finally, the second stage bootloader loads, validates, initializes and executes the kernel.

Every component, which is loaded before the kernel is executed (including the kernel itself), is shipped as an img3 image, which is cryptographically signed and encrypted with Cipher Block Chaining (CBC)-AES-256. Each img3 image contains an Initialization Vector (IV) and key, which are encrypted using a device model specific hardware fused Key Encryption Key (KEK) also called Group Identifier (GID) key. After the signature of the img3 image is verified, the GID key is used to decrypt the corresponding image specific IV and key, which are then in turn used to decrypt the image itself.

The GID key (as well as the UID key) is never exposed to the Central Processing Unit (CPU) directly. Instead, decryption oracle queries are made to a dedicated hardware AES engine, which uses the GID/UID key internally. More precisely, there is a software-based selection of the key slot the Advanced Encryption Standard (AES) engine uses for encryption.

On newer iPhones (starting from the iPhone 4s), the second stage bootloader even disables selection of the GID as key to the AES engine before executing the kernel (enforced

through hardware registers), ruling out any usage by the operating system. Other key slots, such as the one containing the UID key or user supplied keys, can still be selected by the operating system.

### 3.2 iPhone Data Protection and User Authentication Mechanisms

A detailed overview of the iPhone's Data Protection and User Authentication Mechanisms can be found in [App12]. All necessary concepts for this work are described in the following.

Before written to the flash memory, every file is – per default – encrypted with a 256-bit per-file-unique key utilizing the AES engine running in CBC mode. Note that, disabling encryption of the filesystem by the user is not possible. A per-file key itself is wrapped with a key corresponding to a certain protection class and stored in the file's metadata.

The system contains different file classes with various access rights, shortly explained below.

- Files of the *Complete Protection* class can only be accessed while the iPhone is unlocked. Their in-memory keys are discarded after the device has been locked for 10 seconds.
- The *Protected Unless Open* class allows creating new files while the device is locked, but once the file is closed, it cannot be reopened until the user unlocks his device.
- The *Protected Until First User Authentication* class prevents the files from being accessed on a fresh boot until the user unlocks the iPhone for the first time by the passcode.
- Finally, *No Protection* is the default class for all files not assigned to a specific other class. Note that, even in this case, the corresponding files are stored in encrypted form. This prevents attackers from accessing the files by desoldering and dumping the flash memory.

The class keys in turn are wrapped by a key derived by combining the device-specific UID key and the user passcode (if set, except for the *No Protection* class) and stored in a file referred to as the *System Keybag*.

According to Apple's whitepaper the System Keybag is unlocked, i.e., the class keys are unwrapped, by means of Password-Based Key Derivation Function 2 (PBKDF2) processing the user passcode by a Pseudo Random Function (PRF) being AES making use of the UID key [App12]. The PBKDF2 internally iterates the PRF for a high number of times. The iteration count is chosen to lengthen a single unlock attempt to approximately 80 milliseconds, resulting in the iteration count being set to 50,000 in case of the iPhone 4.

By default, iOS limits users to perform only a total of 10 attempts (to unlock the iPhone with passcode) with increasing time delays in between, after which – if enabled – the device wipes the system, rendering all data inaccessible. Note that this is a software-induced restriction which can be bypassed by utilizing known BootROM exploits.

On first setup, the user is prompted to choose a 4-digit numeric passcode (or 6-digit starting from iOS 9) as the default user authentication method. Although iOS offers options to use longer numeric or even alphanumeric passcodes, to the best of our knowledge and based on our personal survey asking 3,864 iPhone users, the majority of users just stays with the default option.

### 3.3 Side-Channel Analysis

Instead of targeting the cryptographic algorithms as in classic cryptanalysis, physical attacks aim at recovering the secrets stored in or processed by the so-called cryptographic device as a particular realization of cryptographic algorithm(s). Side-Channel Analysis (SCA) attacks, as a passive and non-invasive class of physical attacks, have absorbed the lion's share of attention in the scientific community due to their high efficiency. Further, such



attacks leave no sign on their back indicating the device has been compromised. SCA attacks exploit dependencies between physical properties of the implementation and the processed secret data. Next to the execution time [Koc96], other exploitable side channels have been introduced over time, including but not limited to power consumption [KJJ99] and Electro-Magnetic (EM) radiations [GMO01]. In short, nowadays it is well known that a cryptographic device would be vulnerable to various SCA attacks unless the implementation is equipped with countermeasures dedicated to each attack vector.

**Correlation Power Analysis.** In this work, we mainly use CPA, where the measured power/EM traces are correlated to the result of a hypothetical power/EM model over the key-dependent intermediate values of the underlying cryptographic algorithm. This process is conducted in a divide-and-conquer fashion allowing the attacker to recover the secret key in small portions, e.g., byte by byte in case of the AES. As a side note, since Pearson’s correlation coefficient estimates the linear dependency of two random variables, the feasibility of a CPA attack depends on the linear dependency (similarity) of the hypothetical power/EM model to the actual leakage of the attacked cryptographic device. In such a case, the correlation associated to the correct key guess should show a distinguishable distance to that of the other key candidates. Apart from collecting low-noise SCA measurements, the difficulty of CPA attacks indeed lies on choosing an appropriate intermediate value and finding a fitting hypothetical power/EM model [MS16].

**Leakage Assessment.** In this work, we apply the fixed-versus-random t-test [SM15], making use of a statistical test based on the student’s t-distribution. In short, two groups of SCA measurements are collected: 1) when the cryptographic device (with a secret key) is supplied by a fixed input (plaintext in case of encryption) and 2) when random input is given to the device (with the same secret key). The first-order fixed-versus-random t-test examines whether these two groups of SCA measurements are distinguishable from each other through their sample mean (average). If so, it is said that very likely there is an attack which can exploit such a distinguishability to recover the secret. Since the result of a t-test is a confidence level (probability) of the aforementioned distinguishability, the higher the t-statistics is, the higher leakage (stronger distinguishability) is predicted.

## 4 SCA Attacks on iPhone

### 4.1 iPhone Preparation

For the collection of SCA measurements we applied a set of non-permanent soft- and hardware modifications to the iPhone. These modifications are described in the following.

#### 4.1.1 Hardware Preparation

First, we disassembled the iPhone, removed its mainboard from the case, and disconnected all peripherals in order to gain access to the CPU. Afterwards, we removed the metal shield protecting the CPU which enables placing an EM probe directly at the top of the chip. Next, we built a Universal Asynchronous Receiver Transmitter (UART) connector [Ess] utilizing a PodBreakout [pod] connector and an FT232RL USB-to-Serial Breakout Board [FT2]. This yields a connector with two USB cables, one for communicating with the iPhone and one for UART to be connected to a Personal Computer (PC).

Additionally, we removed the volume buttons from the case and connected wires to the volume-down button for easy access. This is done for the purpose of providing a fast and low latency interface to the CPU as these buttons are directly connected to the SoCs GPIO pins.

Afterwards, we dismantled the battery connector, enabling to operate the mainboard with an external DC Power Supply set to 4.0 V (which is slightly above the normal supply voltage of 3.7 V) draining on average about 100 mA during the measurements. Note that this was an ad-hoc choice which is not expected to influence the results of the attack, as the relevant ICs are powered via further voltage regulators.

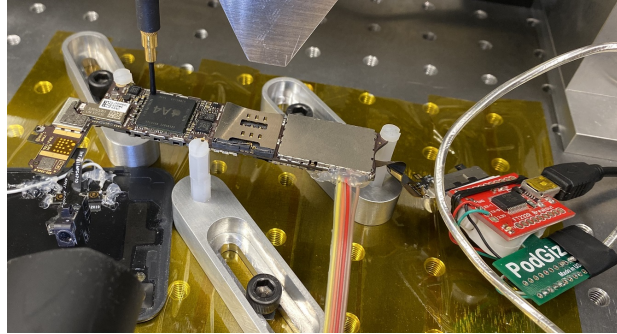


Figure 1: iPhone 4 mainboard mounted on a stage

#### 4.1.2 Payload Execution

As we need arbitrary code execution to perform our measurement, we make use of public exploits. The crucial steps required to enable execution of a custom payload in the second-stage bootloader are described in the following.

As an initial step, the iPhone is forced to boot in DFU mode. Therefore, it has to be first connected to a PC using a USB cable. In order to enter DFU mode, the power button and the home button are pressed simultaneously for 10 seconds. After the power button is released, the home button should still be held pressed for another 15 seconds. The screen stays black when DFU mode is entered successfully.

After the iPhone entered DFU mode, we used the SHatter exploit to disable signature checks in BootROM. To this end, we utilized *ipwndfu* [ax], which is a python tool providing BootROM exploits for several iOS devices. Subsequently, we used *irecovery* [lib], an open source tool for communicating with iOS bootloaders over USB, to transfer a patched first- and second-stage bootloader.

The applied patch disables signature checks in the first-stage bootloader allowing us to execute a modified second-stage bootloader. In the second-stage bootloader – which provides a proprietary recovery console – one of the commands (namely *go*) is redirected in order to force the program flow to jump to the *loadaddress* which is the location in memory where data uploaded over USB is stored. As a result, applying this patch allows uploading and executing custom payloads on the device.

Finally, we again used *irecovery* to transfer a custom payload binary and to interact with the second-stage bootloader recovery console. By executing the previously patched *go* command, we ran the uploaded payload.

#### 4.1.3 Measurement Payload

The custom payload executed in the second-stage bootloader enables querying encryption/decryption of chosen plaintexts/ciphertexts by the on-chip AES engine. Furthermore, it enables us to choose which key slot (GID, UID, or custom) is used by the AES engine. In the following, the structure and functionality of the inserted code is described.

First, the GPIO interface for the volume-down button is reconfigured to act as an output port. This allows driving the corresponding exposed pins high/low by utilizing

Memory Mapped Input/Output (MMIO), i.e., by simply writing a value to a specific address in memory.

Next, the bootloader’s builtin AES routine, responsible for communicating with the hardware AES engine, is patched to set the GPIO pin on high at the start of the encryption/decryption, and on low right after its termination. This configuration enables tabbing the exposed pin and detecting the time instances associated to the activity of the AES engine. Therefore, we could easily use the tabbed signal to trigger the oscilloscope collecting SCA measurements.

Finally, we replaced a recovery console command to enter a custom *measurement* mode. The measurement mode is a custom function which first disables the bootloader’s cooperative scheduler by patching the yield-function to return immediately. Subsequently, it enters an infinite loop which waits to receive a command over UART and executes its functionality accordingly. We developed the custom measurement mode to ease the operations necessary for SCA measurements. Due to the slow communication via UART we configured the measurement mode to limit the UART communications while allowing to collect several SCA measurements. To this end, we followed the concept introduced in [SM15], meaning that the PC sends a few custom commands to the iPhone via UART, the measurement mode configures the AES engine accordingly, generates random input (plaintext/ciphertext) and runs the AES encryption for a certain number of times, and finally sends back a checksum to the PC. During this time the oscilloscope is repeatedly triggered and collects SCA traces. Since the PC and iPhone are synchronized (via the initial commands), the PC calculates the randomly-generated plaintexts/ciphertexts as well and associates them to the traces collected by the oscilloscope. This process greatly accelerates the SCA measurement process.

## 4.2 The Apple A4

Our target device – the iPhone 4 – uses the Apple A4 SoC, which is also used in the iPod Touch fourth generation, the iPad first-generation and the Apple TV second-generation. The Apple A4 provides a 32-bit ARMv7-A CPU manufactured on Samsung’s 45 nm fabrication process [chi10], clocked at 800 MHz (or 1 GHz in case of iPad) with Package on Package (PoP) used to provide 256 MB Random Access Memory (RAM) (or 512 MB for the iPhone 4).

Due to the PoP, the RAM is located on top of the CPU as can be seen in Figure 2. It is indeed impossible to put an EM probe very close to the CPU surface to monitor its direct emanations. Instead, we are only able to put the probe either on top of the packaging or on the Printed Circuit Board (PCB) next to the chip, measuring the emanations associated to the power distribution network around the die.



**Figure 2:** Cross-section of the A4 processor + RAM Package on Package (PoP) (taken from iFixit A4 teardown [ifi10])

## 4.3 Measurement Setup

We use a **Langer EMV-Technik RF-B 0,3-3** EM probe, which has a flat head with a diameter of 2 mm allowing to capture frequencies in the range of 30 MHz to 3 GHz. The

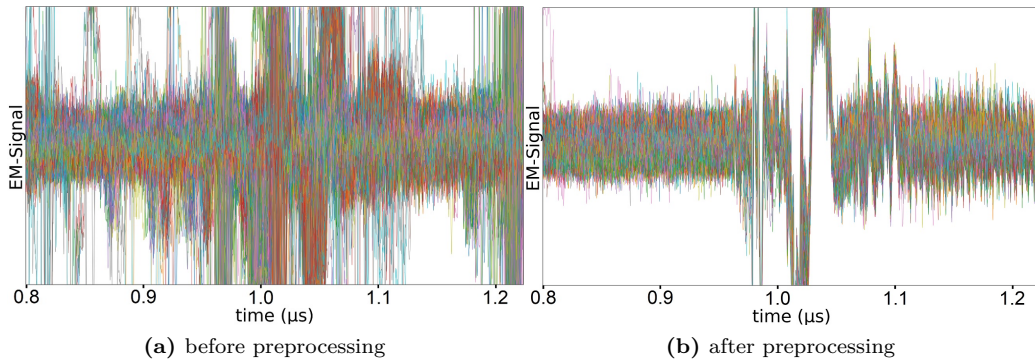
probe is connected to a **Langer EMV-Technik PA 303 SMA** amplifier (with similar bandwidth), which amplifies the EM signal by 30 dB, before the signal is monitored by the oscilloscope.

We employed a **Teledyne LeCroy WaveRunner 8254M** with 2.5 GHz bandwidth for monitoring the signals and recording the SCA traces. The traces have been sampled at a sampling rate of 40 GS/s, which is the machine's maximum capacity.

#### 4.4 Preprocessing of the Traces

Figure 3a shows a part of the superimposition of 500 collected EM traces, where we detect a heavy misalignment, complicating any straightforward statistical analysis. Although we configured the measurement mode to run on a single-threaded core in bootloader context, it can clearly be seen that the trigger signal, which we provided to indicate the start and end of encryption/decryption is not fully synchronized to the activity of the AES engine. It actually implies that the code running on the CPU core does not work synchronously either with the co-processors or with the IO peripherals, which controls the GPIO pin we used for triggering the oscilloscope.

As a consequence, one part of our preprocessing tries to align the traces. Here, we realized that the misalignment often appears in groups, i.e, multiple traces are shifted by a similar offset. Thus, our main approach is to find those clusters and coarsely align them to one group by shifting them so that the strongest peaks overlap. Afterwards, for a more fine grained alignment, we chose an arbitrary *reference trace* and align the other traces by an appropriate temporal offset so that they match as closely as possible to the reference trace. We used the minimum euclidean distance as the metric to find the best matching offset between two traces.



**Figure 3:** Superimposition of 500 EM traces before and after preprocessing

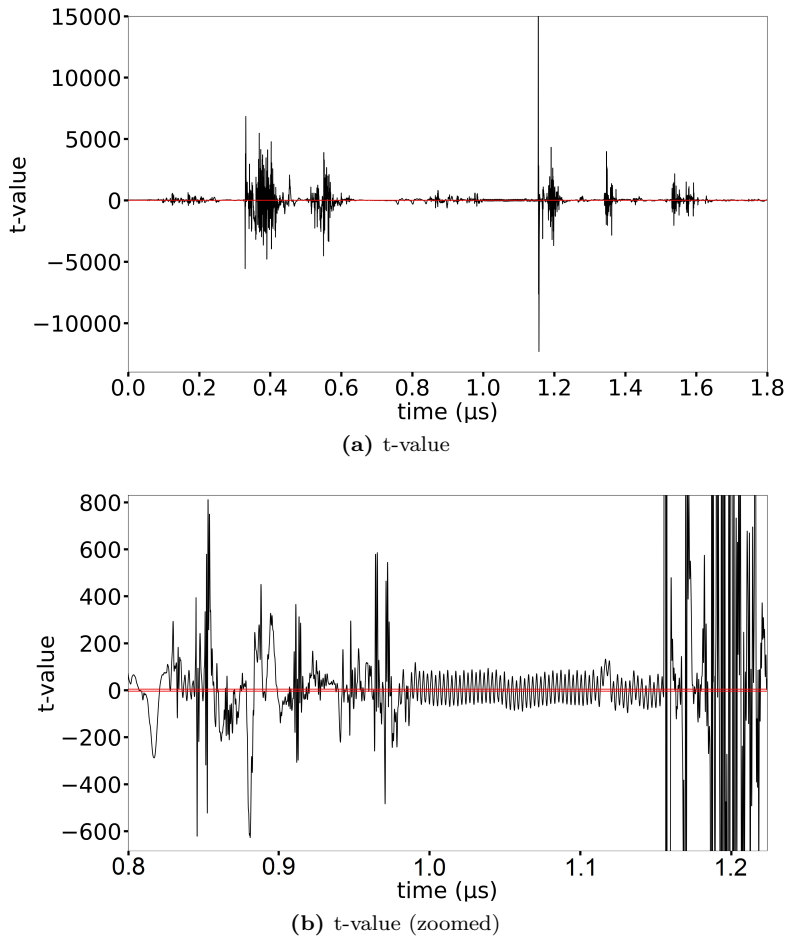
As after alignment, a significant amount of traces still showed some noisy peaks going out of bounds, we additionally filtered out traces whenever these peaks occurred during the time interval on which we performed our analysis, thereby discarding around 20% of the collected traces. The result after applying our alignment and filtering process on those 500 traces can be seen in Figure 3b.

As a side note, we noticed that the jitter and misalignment is greatly reduced when decrypting multiple blocks consecutively. Hence, in our attacks and analyses we always collected the traces associated to decrypting 8 blocks in CBC mode.

#### 4.5 Leakage Assessment

First, we performed the fixed-versus-random t-test (see Section 3.3) on the aligned traces when the EM probe was placed arbitrarily on the SoC. We collected 10,000,000 traces

while the input (ciphertext for the decryption function) was randomly interleaved between a fixed value and a random input.



**Figure 4:** Non-specific t-test

Figure 4a shows the corresponding t-value over the time covering the decryption of 8 blocks of AES in CBC mode. Very large t-values can be seen in several portions of the traces. Most notably, the period between  $0.3 \mu\text{s}$  and  $0.6 \mu\text{s}$  corresponds to the communication between the CPU and the AES engine through the dedicated Input/Output (I/O). The chunks around  $1.2 \mu\text{s}$ ,  $1.4 \mu\text{s}$  and  $1.6 \mu\text{s}$  are predicted to be relevant to the similar communication in the reverse direction, i.e., AES engine to the CPU. We assume that the decryptions take place in between  $0.8 \mu\text{s}$  and  $1.2 \mu\text{s}$ , which still show a considerably-high t-value (see Figure 4b). Hence, we concentrated on this period to conduct the attacks.

## 4.6 Power Model

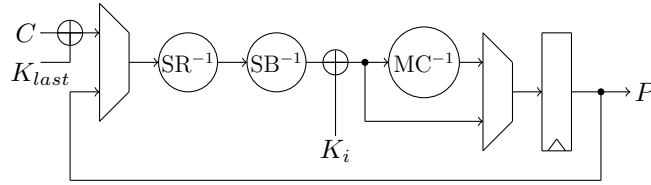
The iPhone AES engine allows to specify a user key for its operation, thus we search for an appropriate power model in a known-key scenario, i.e., we collected traces for which we know the underlying key. This way we are able to easily examine different hypothetical power models since all cipher's intermediate values are known to us.

Based on the information from Apple's whitepaper describing how the AES engine is used, we made certain assumptions about how the AES hardware might have been designed. We know that the AES engine needs to be fast since during the operation of the

iPhone, key derivation as well as encryption/decryption of files are performed quite often. A slow AES engine would lead to serious efficiency penalties. It should also support both encryption and decryption of different variants of AES with 128-, 192- and 256-bit key. Furthermore, due to the way data protection is designed for iPhones, we know that the AES key is changed frequently (each file has a separate key).

Therefore, we assume a round-based implementation, as it is a good trade-off between flexibility, performance and physical area on the SoC. More precisely, our assumption is that the AES engine performs a cipher round in a clock cycle.

Based on these assumptions, we considered different design architectures for the AES decryption, and examined various hypothetical models over different intermediates values. Examples include Hamming Weight (HW) of the cipher state after each round operation (e.g., SubBytes (SB), ShiftRows (SR), AddRoundKey, and MixColumns (MC)) and Hamming Distance (HD) between consecutive cipher states for each aforementioned round operation.



**Figure 5:** Gussed design architecture for the round-based AES decryption module

Figure 5 depicts the gussed round-based design architecture, for which we could observe high correlation by the aforementioned power models. We determined the best fitting power model as the HD of the consecutive 128-bit values stored in the state register (see Figure 5), if we follow the operations inversely, i.e., from the last round of decryption. The last value stored in the state register is the plaintext  $P$ , and the second-last one is  $SR \circ SB(P \oplus K_1)$  since the MixColumns' inverse ( $MC^{-1}$ ) is omitted in the last round in this design architecture. Therefore, the HD of the state register in the last decryption round can be written as

$$HW(SR \circ SB(P \oplus K_1) \oplus P), \quad (1)$$

where the first round key is denoted by  $K_1$ . The same model for the second-last round yields

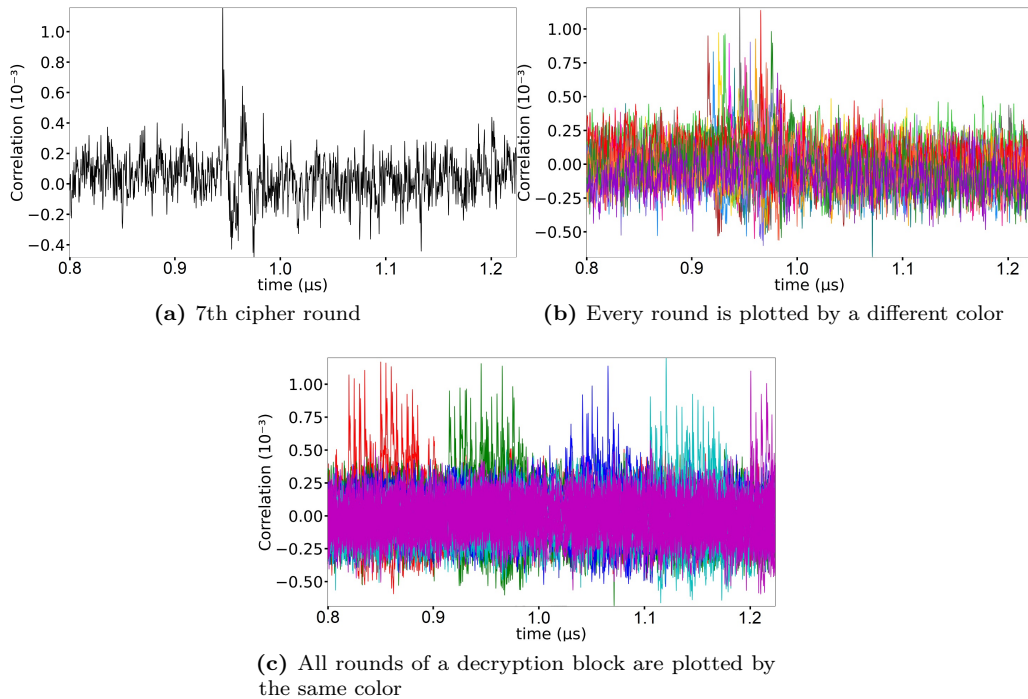
$$HW\left(SR \circ SB(K_2 \oplus MC \circ SR \circ SB(P \oplus K_1)) \oplus SR \circ SB(P \oplus K_1)\right), \quad (2)$$

where  $K_2$  denotes the second round key. The same model can similarly be derived for the other cipher rounds. Note that since we focus on the last rounds of decryption, we write the equations over the plaintext and using the encryption operations instead of their inverse.

The result of correlating 80,000,000 measured and aligned EM traces with the aforesaid power model (when the AES engine's inputs are selected randomly) are shown in Figure 6. The cipher rounds can be easily detected as depicted in Figure 6b. We further repeated this procedure for all encryption blocks within the trace. As shown in Figure 6c, we can clearly distinguish the distinct time periods in which an AES operation is performed.

#### 4.6.1 Full-Chip Scan

After we found a promising hypothetical power model, we tried to optimize the probe's position on the SoC. We divided the SoC surface (which is around 7.3 mm square) into a



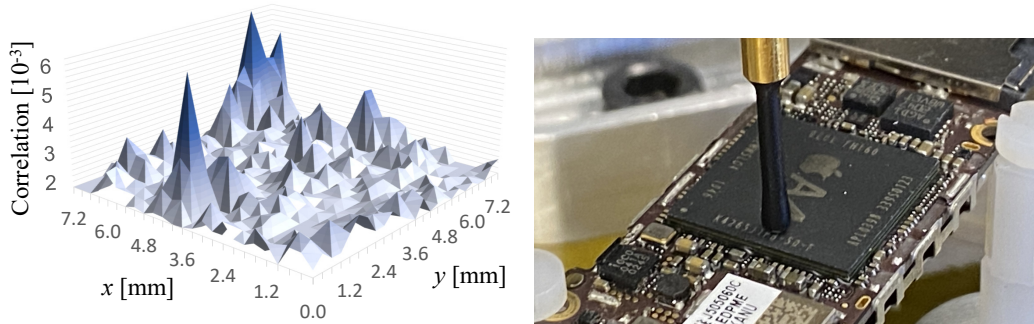
**Figure 6:** Multiple correlation traces associated to the selected HD power model, using 80,000,000 traces

grid of  $25 \times 25$  spots, at each of which we collected 150,000 EM traces. After aligning the traces at each spot individually, we estimated the correlation of the intermediate values using the above-explained HD power model for all cipher rounds and for all decryption blocks. Maximum resulting correlations for each spot are shown by a 3D heatmap on the left-hand side of Figure 7. In case a strong signal stands out at a certain spot, neighboring positions are expected to have a similar correlation value as well. As shown by the 3D heatmap, maximum correlations belong to the cases where the probe is placed at the border of the SoC. This is indeed in line with our expectation with respect to its PoP technology, as explained in Section 4.2. Based on this experiment, we identified the position  $(x, y) = (4.1 \text{ mm}, 1.1 \text{ mm})$  as the most suitable spot. We have examined other spots with high correlation as well, but the traces at those positions contained more noise compared to those measured at the selected spot. For the key-recovery attacks, explained in the following, we placed the EM probe at this position, shown on the right-hand side of Figure 7, and collected 500,000,000 traces when the AES engine was supplied by random inputs.

#### 4.6.2 Key-Recovery Attacks

All above given results were based on the HD model over consecutive 128-bit cipher-intermediate values. In order to perform an attack, we need to consider a smaller portion to decrease the attack complexity, i.e., being able to search for a small part of the key e.g., a byte. This is trivially achieved by taking the HD of a byte of the same intermediate value. More precisely, we refer to Equation (1), where by guessing an 8-bit portion of  $K_1$ , the HD of the corresponding 8-bit consecutive intermediate value can be estimated. Following this process, we can recover the first 128-bit round key in a byte-by-byte fashion.

Since the underlying AES engine realizes the AES-256 function and the targeted UID



**Figure 7:** 3D heatmap of the SoC plotting highest correlation in time using the selected HD power model and 150,000 traces at each spot (left) and probe position (right)

is a 256-bit key, we need to extend the attack to one more round, i.e., the round before the last decryption round. As given in Equation (2), knowing the first round key  $K_1$ , HD of the consecutive 8-bit values can be estimated by only guessing an 8-bit portion of the second round key  $K_2$ . Therefore, the same attack, i.e., with complexity of  $2^8$  for each byte of  $K_2$  can be conducted at the second to last round.

Figure 8 shows the result of a couple of attacks targeting different key bytes at either the last or the second last decryption rounds. Notably, we required a large number of aligned traces to reveal the correct key guesses. For some key bytes, we needed around 30,000,000 traces, and in some other cases this number reached 270,000,000. As the AES engine is run in CBC mode, each recorded EM trace contains a series of single AES decryptions; in case a block did not yield a distinguishable correlation, a different one was considered. This helped us to limit the number of required traces since not all decryption blocks in a noisy trace are affected.

In short, we have conducted three sets of attacks: one when the key was known to us (user supplied key), one to recover the UID key and the last one to reveal the GID key. For each of these cases, which led to successful key recovery, we required not more than 300,000,000 traces. This is much more than what has been reported in literature with respect to the SCA attacks on unprotected cryptographic implementations, e.g., [MS16, EKM<sup>+</sup>08]. We predict that either the implementation contains a form of Differential Power Analysis (DPA) countermeasure or this high number of required traces is due to the high noise level and low signal amplitude. As stated, the EM probe could not be placed close to the CPU die and we encountered different noise sources in our measurements. Nevertheless, our results (despite a high number of required measurements) in fact confirm that such SCA attacks can still be considered as serious threats even to extremely compact embedded systems fabricated with nano-scale process technologies and running at a high clock frequency<sup>1</sup>.

In summary, data acquisition of plaintexts, ciphertexts and traces for 500,000,000 EM measurements took about two weeks in total, while conducting all attacks to fully recover a 256-bit AES key using a machine with 40 CPU cores, 64 GB of memory and 2 NVIDIA RTX 2080 TI took around one additional week. Note that we expect the required time to be shorter when performing the attack multiple times on further devices of the same type due to the experience obtained during each device's analysis and the natural optimization of the measurement procedure. We are also confident that improvements of the measurement setup are possible which may reduce the data and time complexity of this attack.

<sup>1</sup>Assuming a round-based implementation, based on our SCA measurements and correlation peaks identifying consecutive cipher rounds in Figure 6, we conclude that the AES engine is supplied by a clock at a frequency of around 200 MHz.



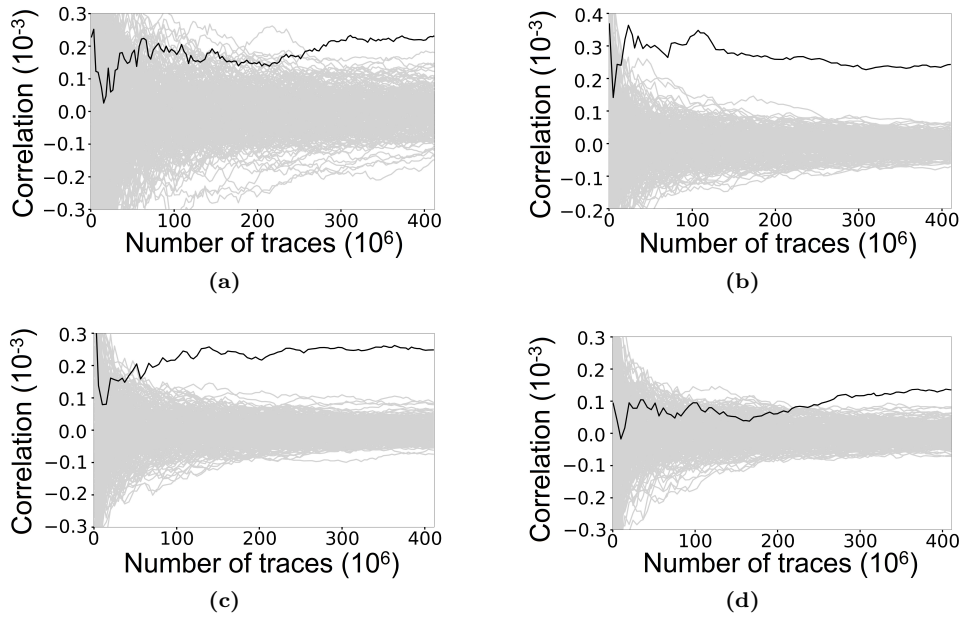
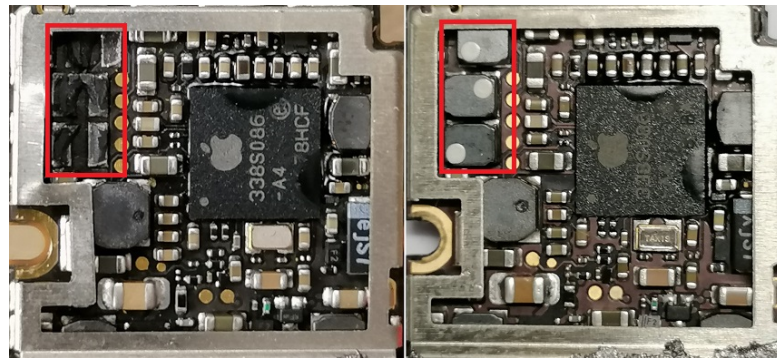


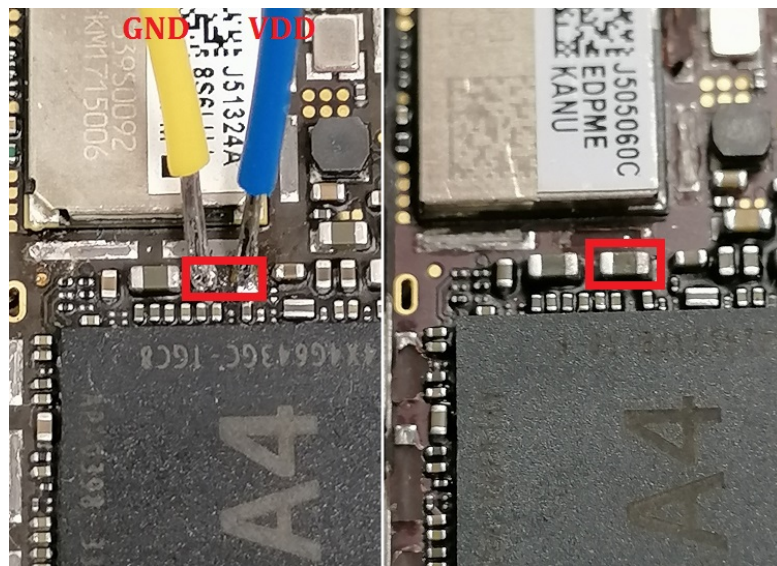
Figure 8: Exemplary CPA attack results

#### 4.6.3 Power vs. EM

Given the large number of measurements required to recover the key of the AES engine via EM measurements, it is fair to wonder whether the distance between the probe and the active area of the die was simply too large to capture the radiated information in maximum quality. Since the PoP packaging prevents us from placing the EM probe any closer to the actual AES core, we decided to also investigate the power consumption of the A4 processor. Conceptually, power measurements do not require such a close physical proximity to the computing cells. Therefore, we placed a  $1\Omega$  shunt resistor in the VDD path of the core power supply of the A4 processor. Of course, the core power supply of the CPU is not directly accessible, e.g., via the battery connector. Instead, it is generated on the PCB via a Buck converter circuit from the main power supply. Measuring the power consumption in front of this converter (i.e., measuring in the main power supply at the converter’s input) would yield no successful analysis as any small instantaneous voltage drop on its output side (i.e., in the core power of the chip) gets quickly compensated by the internally stored charges and only the charging cycle of the converter could be seen on its input. Therefore, after we had identified the position of the Buck converter circuit on the PCB, we removed its inductors, effectively cutting the core power supply open, see Figure 9a. Afterwards, we removed one of the larger capacitors on the PCB which we previously identified as a smoothing capacitor for the core voltage. Then, we soldered two cables to the SMD pads of the (now missing) smoothing capacitor and powered the core of the A4 processor through these pads via an external DC Power Supply at 1.35 V, see Figure 9b. Please note that the pictures in Figure 9 show two different devices and not the same board before and after modification. Although we destroyed the inductors of the Buck converter during the removal process, it is generally possible to perform both adaptations of the PCB carefully enough to keep all pieces intact in order to revert the changes later on. We also need to mention here that the iPhone struggled to boot after altering the PCB in the described manner. However, after patching the bootloader to operate in a reduced power mode we did not experience any bootloader crashes and could



(a) removed inductors (left) and original PCB (right)



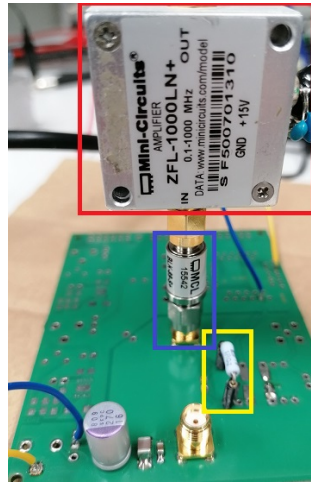
(b) artificial core power supply (left) and original PCB (right)

**Figure 9:** PCB adaptations required for measuring the CPU's core power.

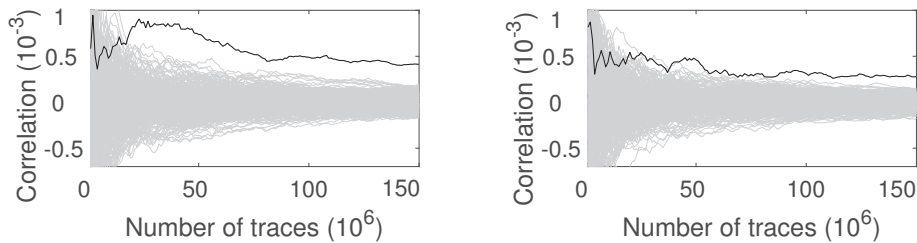
perform the desired measurements. The reduced power mode also leads to decreased operating frequencies and therefore may be beneficial for physical adversaries anyway.

As mentioned before, a  $1\Omega$  shunt resistor needs to be placed in the VDD path of the power supply. This was realized with an auxiliary board connected between the external power supply and the iPhone, see Figure 10. We employed a DC Blocker (BLK-89-S+ from Mini-Circuits<sup>2</sup>) to remove the DC shift and an AC amplifier (ZFL-1000LN+ from Mini-Circuits) to increase the amplitude of the measured signal. Then, we measured the voltage drop across the CPU's core via a coaxial cable connected to the amplifier's output. We have recorded the measurements using our digital oscilloscope configured to a bandwidth limit of 1 GHz and a sampling rate of 2.5 GS/s. Similar to the EM measurements, we could not identify any AES-like sequence of power peaks in the traces. Even more problematic was the absence of any communication or IO peaks in the traces which could have been used to achieve the same re-alignment that was previously detailed for the EM analysis. Hence, we had to perform the attacks on the raw traces without any pre-processing. Nevertheless, Figure 11 shows two successful recoveries of different key bytes via CPA using the same power model as for the EM analysis. Here, about

<sup>2</sup><https://www.minicircuits.com>



**Figure 10:** Auxiliary board with AC amplifier (red), DC blocker (blue) and  $1\Omega$  shunt resistor (yellow).



**Figure 11:** Exemplary CPA attack results exploiting the power consumption.

20 and 75 million traces were required to isolate the correct key candidates from the incorrect ones. However, attacks on other key bytes required up to 200 million traces for an unambiguous recovery. In that regard, we can conclude that the attacks exploiting the power consumption are hardly any more effective than the EM attacks, especially when considering the additional effort to modify the PCB. Yet, both side channels can be used in practice to successfully recover the keys processed by the hardware AES engine.

## 4.7 Reversing the Modifications

As described in Section 4.1.1, for collecting the EM traces, the case of the iPhone has to be opened in order to expose the battery, the mainboard, and the CPU. The battery is dismantled and the volume buttons are removed from the case. Note that the necessary modifications can be made on a separate set of hardware allowing the target device to remain mostly unmodified. For example, wires can be soldered to a spare volume-button-cable, which is then (in place of the original part) temporarily attached to the mainboard using the available connector. Furthermore, the on-board battery connector does not need to be modified, when instead a modified battery cable is used. In addition, the metal shielding has to be removed. As long as its mount on the board is left in place, this can also simply be clipped back on afterwards. For the power measurement described in Section 4.6.3, some inductors and a large capacitor had to be removed from the PCB. These elements can be soldered back onto the board after performing the attack. As a result, the modifications necessary to perform the attacks are entirely reversible, enabling

an attacker to completely hide the malicious key recovery from the victim.

## 4.8 Possible Countermeasures

In order to protect future devices from attacks similar to the one presented in this work, several suitable options are available ranging from software protections over architecture-level defenses to classical DPA countermeasures. For the newer iPhone series (from 5s onward), Apple itself has already applied an additional architecture-level barrier to protect the most sensitive hardware-fused keys from extraction, namely by introducing the Secure Enclave, a dedicated processor unit, and prohibiting usage of the SEPUID (used for the entanglement with the passcode) from the application processor [App21]. This way, a software exploit on the application processor is not sufficient to get oracle access to the relevant crypto engine and is hence not sufficient to perform a chosen-plaintext SCA attack.

Naturally, possible protection mechanisms of the crypto engine itself include classical DPA countermeasures like *masking* and *hiding* which aim to either split the sensitive information into a number of shares using randomly chosen masks which shifts leakages to higher orders, or reduce the signal-to-noise ratio and make the measured traces hence less informative [MOP07]. According to [App21] and beginning with Apple's A9 processor, the company has also started to integrate DPA countermeasures into their designs, although no further details on the kind of countermeasures that were implemented are given. Further possible countermeasures could include the introduction of tamper protection, making it harder to physically modify the device without the victim's notice [App21].

## 5 Offline PIN Recovery

Up to this point, we recovered the UID key by performing an SCA attack. We now show that being in possession of this key, the brute-force attempts completely decouples from the device, enabling a highly parallelized search for the user's passcode by a GPU cluster.

### 5.1 Dumping the System Keybag

As discussed in Section 3.2, the System Keybag contains wrapped class keys protected by the UID key tangled with the user passcode via a key derivation function. For each passcode guess, the key derivation function should be executed to calculate the guessed derived key. Afterwards, the correctness of the guessed derived key is examined by attempting to unlock every class key from the keybag. If every class key could be successfully unlocked, the correct passcode is found.

In order to extract the System Keybag from the device, we used msftguy's *ssh-rd* tool from [msf]. It is a tool written in Java, which automatically downloads the iPhone firmware from Apple's servers; extracts, decrypts and patches bootloaders as well as the kernel, and creates a ramdisk providing an SSH server, which is then booted on the device. As this is similar to one-time booting a live Linux distribution from USB on a computer, no modifications are made to the filesystem. Note that it does not harm if the booted ramdisk uses a possibly older version of iOS than the installed one.

Next, we established an SSH connection to the device in order to download the System Keybag which is stored at `/private/var/keybags/systembag.kb` on normal boot or `/mnt/keybags/systembag.kb` on a ramdisk boot (if we mount the user data partition to `/mnt`).

### 5.2 Highly Parallelized Passcode Recovery utilizing GPUs

For the purpose of recovering the user's passcode, we implemented a parallelizable program in *OpenCL* whose speed scales with the number of GPUs of a single host, as well as with

the number of hosts. Utilizing OpenCL offers independence from the underlying hardware, i.e., our implementation can be executed on several architectures including NVIDIA, AMD, Intel, ARM Mail or any other hardware supporting OpenCL. Our implementation is split into

- (1) parsing the System Keybag,
- (2) generating the passcode batch,
- (3) pre-processing the passcode batch,
- (4) deriving the key based on the passcode batch, and
- (5) verifying the derived key batch.

Step (1) and (5) are performed on CPU, since parsing the keybag needs to be performed only once and unwrapping class keys – following RFC 3394 – is not computationally intensive and can be stopped once a single key failed to unwrap. Note that, in order to verify the correctness of the derived key batch, we considered unlocking all 10 class keys from the System Keybag.

The other steps are performed on GPU with the batch size being calculated as  $\text{GPU\_workgroup\_size} \times 64 \times \text{GPU\_factor}$  with the `GPU_workgroup_size` being 1024 for our NVIDIA RTX 2080 TI GPUs and the `GPU_factor` being manually set to 64 in order to balance performance and processing time per batch loaded onto the GPU. For our setup, this results in a batch containing  $2^{22} = 4,194,304$  passcode guesses which takes about 14 minutes to be processed. It is possible to further decrease the `GPU_factor` to process a smaller number of passcode guesses, i.e., when only searching for a 4-digit passcode in a single batch. However, these values are determined as optimal when searching for a number of passcode guesses higher than those fitting into a single batch.

Step (2) generates a batch of passwords to be stored in memory. In our program, we limited the generation of passcodes to numeric ones, while its extension to cover alphanumeric passcodes is straightforward. However, we would like to note that finding alphanumeric passcodes becomes more efficient when combined with dictionary attacks or other password-search techniques as described in [AHW18], which are out of the scope of this work.

Next, step (3) performs PBKDF2 to compress the arbitrarily length string (i.e., passcode) to a sequence of 32 bytes. Unlike described in [App12], the key derivation function is not implemented as PBKDF2 with AES as the PRF. It rather consists of a single iteration of the regular PBKDF2 algorithm with SHA1-HMAC as PRF followed by Apple’s custom Key Derivation Function (KDF).

Step (4) performs Apple’s custom KDF which consists of the PBKDF2-derived 32-byte user key, the UID key, an iteration count, and an internal 32-byte state (as reverse engineered by Sogeti in `ramdisk_tools/AppleKeyStore_kdf.c` in [sog11a]). First, the internal state is initialized with the user key. Afterwards, the following is executed in a loop: The IV (which is initially set to zero) is salted with the current iteration count and XORed to the user key, which is then encrypted utilizing AES-128 in CBC mode keyed by the UID key. The output is then XORed to the state and the last block (of two blocks in total) is used as the new IV. This is repeated as many times as specified by the iteration count, before the state is returned as output (50,000 times in case of our target iPhone 4).

Since this is the computationally most intense operation, we implemented a bitsliced version of AES which processes 64 blocks in parallel. This is even further parallelized on multiple GPU threads, thus computing 64 work groups of size 1024, each of which computes 64 blocks at a time, resulting in a total of 4,194,304 blocks being processed in parallel.

Finally, step (5) is performed on multiple CPU threads and uses the previously computed keys to unlock all class keys from the System Keybag. Only if all class keys are successfully unlocked, the corresponding passcode is considered to be the correct one. As most of the

time the unlocking of the first class key from the System Keybag fails, this step usually terminates rapidly.

### 5.3 Results

Table 1 shows a comparison between the required time to find numeric passcodes in the worst-case scenario for different passcode lengths when performing the brute-force search on device and on a GPU cluster. Given the ability to execute arbitrary code, short numeric passcodes can be reasonably searched on-device. Assuming the verification of a single passcode guess takes 80 ms as described in [App12], finding 6-digit passcodes requires 22 hours (in worst-case) if verification is performed on device. Our practical experiments with Sogeti's `iphone-dataprotection` toolkit from [sog11a] on an iPhone 4 show that verifying 2,000 passcodes takes 6 minutes (180 ms per attempt). This means that the actual numbers given in Table 1 for the on-device search are higher by a factor of 2.25. According to Sogeti, the time to crack a passcode may vary depending on the device [sog11b]. It should be noted that Elcomsoft reported 73.5 ms per attempt on an iPhone 5 which is slightly faster than the stated 80 ms. Our work decouples passcode search from hardware limitations of the target device and enables performing the corresponding search programs on arbitrary platforms. Using a single NVIDIA RTX 2080 TI, we already achieved a speedup by a factor of 380, making it possible to search for a 10-digit passcode in a reasonable time by simply utilizing a 2-year-old gaming setup. Employing a GPU cluster with 8 instances of NVIDIA RTX 2080 TI, accelerates the search even further. With this advanced setup, searching for an 11-digit passcode would take around 30 days in the worst-case scenario while renting these resources would cost around 2000 EUR. As this scales linearly, the worst-case time to find longer passcodes using more GPU instances can be easily estimated. Overall, taking into account the time required to perform the SCA attack in order to extract the UID key (see Section 4.6.2), our attack outperforms the trivial on-device search approach if the target numeric passcode is at least 8 digit long.

Table 1: Worst-case passcode search time

digits	iPhone	RTX 2080 TI	8×RTX 2080 TI
4	13 minutes	2 seconds	< 1 second
6	22 hours	3 minutes	26 seconds
7	9 days	35 minutes	4 minutes
8	92 days	5 hours	43 minutes
9	925 days	58 hours	7 hours
10	25 years	24 days	3 days
11	253 years	243 days	30 days
12	2536 years	2439 days	304 days

## 6 Applicability to Newer iPhone Series

Generally, this attack is applicable to newer iPhone series as well. Without major changes the same procedure can be performed on iPhone 4s and iPhone 5/5c, when using `checkm8` [ax] instead of `SHAtter`.

Starting with the iPhone 5s, the iPhone features a Secure Enclave coprocessor (SEP), which is responsible for data protection. Instead of using the Application Processor (AP) UID key, the SEP has a separate UID key (SEPUID), which is entangled with the user passcode for file encryption. Thus, to perform the same analysis, it is required not only to get code execution on the AP, but also on the SEP.

For the iPhone 5s, iPhone 6, iPhone 6s, iPhone 7, iPhone 8 and iPhone X (and all other devices with the same CPU), vulnerabilities in both, the AP BootROM and SEP BootROM are publicly known (namely `checkm8` and `blackbird` [Xu]) providing a way to execute arbitrary code with the highest possible privileges on the AP and the SEP. Therefore, it is generally possible to create payloads with similar capabilities as described in Section 4.1.3 to be executed on the SEP rather than the AP. The corresponding exploits for such vulnerabilities are not publicly available for all listed iPhone generations yet. However, at least for the iPhone 7 there exists a publicly available tool called `checkra1n` [aea], which exploits the known vulnerabilities to gain code execution on the SEP. We would like to mention that we have already initiated follow-up research to examine if a similar SCA attack is possible on iPhone 7 and later. In our initial attempts, we were able to gain code execution on the SEP using `checkra1n` and to query the SEP's AES engine to encrypt chosen data. Although this allows collecting SCA measurements from newer hardware, it is hard to predict whether extracting the UID/SEPUID key using the SCA measurements would be similar to (or harder/easier than) that on the iPhone 4, especially since Apple claims to have introduced DPA countermeasures to protect the hardware AES engines starting from the A9 processor generation, i.e., from the iPhone 6s series onward [App21].

## 7 Conclusions

Utilizing public software exploits for known vulnerabilities in order to enable oracle access to the AES engine, we extracted both hardware fused 256-bit AES keys, namely the UID and the GID of an iPhone 4 through Side-Channel Analysis (SCA) attacks processing the Electro-Magnetic emanation and power consumption of the AES engine embedded on the underlying A4 processor. Independent of the implications of our attacks, to the best of our knowledge, no successful SCA attack on an iPhone has been reported in academic literature so far. We, for the first time, presented the success of a corresponding key recovery process despite a compact System on Chip with PoP packaging. Although we need a large amount of traces to recover the complete AES keys, we showed that, if the software barriers are overcome, i.e., there are exploits available to execute custom code on the device, the model of a physical attacker being able to query arbitrary many chosen-plaintext encryptions (or chosen-ciphertext decryptions) is absolutely realistic, emphasizing the need for sufficient protection against SCA – even for hundreds of millions of traces.

Having the UID key in hand, it becomes possible to conduct offline brute-force attacks recovering the user's passcode. Using a highly parallelized GPU implementation, we established a scalable method to highly increase the performance of the passcode search procedure. As the performance linearly scales with the number of utilized GPUs, the search time can be shortened depending on the value of the data versus the budget. We showed that a 10-digit numeric passcode can be revealed in a reasonable time employing a common gaming setup, while a large-scale adversary might even be able to cover 12-digit numeric passcodes.

A possible remedy would be to choose stronger passcodes (either numeric or alphanumeric); however that may result in user inconvenience, as the passcode needs to be entered once in a while despite biometric authentication. We stress that this vulnerability cannot be mitigated on affected devices via a software update, since the leakage originates from the integrated AES engine. In short, our attack emphasizes the need for implementing sophisticated countermeasures against physical attacks, particularly against SCA attacks, for mobile devices, even when the device is a highly compact embedded system and the adversary's knowledge is restricted to a blackbox model.

New software exploits may allow the adversary to interact with the AES engine on the newer iPhone generations. This enables SCA measurements to be collected from such devices and key-recovery attacks to be examined. Trivially, investigating the applicability

of such attacks is among our future works.

## Acknowledgments

We initiated a communication with Apple in a responsible disclosure about our findings on 5 Oct 2020. We would like to thank Apple for their support and kind communication during this process. The work described in this paper has been supported in part by the German Research Foundation (DFG) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

## References

- [aea] @argp et al. checkra1n exploit. [Link](#) [Online; accessed on 16-March-2021].
- [AHW18] Sudhir Aggarwal, Shiva Houshmand, and Matt Weir. New Technologies in Password Cracking Techniques. In *Cyber Security: Power and Technology*, pages 179–198. Springer, 2018.
- [App12] Apple. *iOS Security Guide*, May 2012.
- [App21] Apple. *iOS Security Guide*, February 2021.
- [@ax] @axi0mX. Github: ipwndfu. [Link](#) [Online; accessed on 18-September-2020].
- [BFMT16] Pierre Belgarric, Pierre-Alain Fouque, Gilles Macario-Rat, and Mehdi Tibouchi. Side-Channel Analysis of Weierstrass and Koblitz Curve ECDSA on Android Smartphones. In *CT-RSA 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 236–252. Springer, 2016.
- [chi10] Chipworks Confirms Apple A4 iPad chip is fabbed by Samsung in their 45-nm process, 2010. [Link](#) [Online; accessed on 22-September-2020].
- [EKM<sup>+</sup>08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoqCode Hopping Scheme. In *CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 203–220. Springer, 2008.
- [elc11] Slides: Evolution of iOS Data Protection and iPhone Forensics: from iPhone OS to iOS 5, August 2011. [Link](#) [Online; accessed on 22-September-2020].
- [Ess] Stefan Esser. iPhone UART cable. [Link](#) [Online; accessed on 30-October-2020].
- [FT2] FT232RL USB to Serial Breakout Board - robotshop. [Link](#) [Online; accessed on 30-October-2020].
- [GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In *CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.
- [Goa20] Laurence Goasduff. Gartner Says Global Smartphone Sales Declined 20% in First Quarter of 2020 Due to COVID-19 Impact, 2020. [Link](#) [Online; accessed on 18-September-2020].
- [ifi10] Apple A4 teardown, 2010. [Link](#) [Online; accessed on 22-September-2020].
- [int15] Differential Power Analysis on the Apple A4 Processor. *The Intercept*, 2015. [Link](#) [Online; accessed on 23-March-2021].



- [KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
- [KKMP09] Markus Kasper, Timo Kasper, Amir Moradi, and Christof Paar. Breaking KeeLoq in a Flash: On Extracting Keys at Lightning Speed. In *AFRICACRYPT 2009*, volume 5580 of *Lecture Notes in Computer Science*, pages 403–420. Springer, 2009.
- [Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
- [lib] libimobiledevice. GitHub: libirecovery. [Link](#) [Online; accessed on 18-September-2020].
- [MBG<sup>+</sup>20] Philipp Markert, Daniel V. Bailey, Maximilian Golla, Markus Dürmuth, and Adam J. Aviv. This PIN Can Be Easily Guessed: Analyzing the Security of Smartphone Unlock PINs. In *Symposium on Security and Privacy, SP 2020*, pages 286–303. IEEE, 2020.
- [MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.
- [MS16] Amir Moradi and Tobias Schneider. Improved Side-Channel Analysis Attacks on Xilinx Bitstream Encryption of 5, 6, and 7 Series. In *COSADE 2016*, volume 9689 of *Lecture Notes in Computer Science*, pages 71–87. Springer, 2016.
- [msf] msftguy. GitHub: ssh-rd. [Link](#) [Online; accessed on 18-September-2020].
- [NVI18] NVIDIA. *Geforce RTX 2080 Ti*, 2018. [Link](#) [Online, access 21-March-2021].
- [OP11] David Oswald and Christof Paar. Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World. In *CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 207–222. Springer, 2011.
- [ORP13] David Oswald, Bastian Richter, and Christof Paar. Side-Channel Attacks on the Yubikey 2 One-Time Password Generator. In *RAID 2013*, volume 8145 of *Lecture Notes in Computer Science*, pages 204–222. Springer, 2013.
- [pod] PodBreakout - sparkfun. [Link](#) [Online; accessed on 30-October-2020].
- [SB15] Jeremy Scahill and Josh Begley. The CIA Campaign to Steal Apple’s Secrets. *The Intercept*, 2015. [Link](#) [Online; accessed on 21-March-2021].
- [Sko16] Sergei Skorobogatov. The bumpy road towards iPhone 5c NAND mirroring. *CoRR*, abs/1609.04327, 2016.
- [SM15] Tobias Schneider and Amir Moradi. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In *CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.
- [sog11a] iPhone dataprotection toolkit, 2011. [Link](#) [Online; accessed on 22-September-2020].
- [sog11b] Slides: iPhone data protection - HackInTheBox Amsterdam, 2011. [Link](#) [Online; accessed on 22-September-2020].

- [SRH16] Sami Saab, Pankaj Rohatgi, and Craig Hempel. Side-Channel Protections for Cryptographic Instruction Set Extensions. *IACR Cryptol. ePrint Arch.*, 2016:700, 2016.
- [VMC19] Aurélien Vasselle, Philippe Maurine, and Maxime Cozzi. Breaking Mobile Firmware Encryption through Near-Field Side-Channel Analysis. In *ASHES@CCS 2019*, pages 23–32. ACM, 2019.
- [Xu] Hao Xu. Attack Secure Boot of SEP. Mobile Security Conference (MOSEC) 2020. [Link](#) [Online; accessed on 28-September-2020].

# Chapter 4

## Formal Verification of Masked Hardware Circuits

*In this chapter, we present all peer-reviewed publications accumulated during this thesis that relate to formally verifying SCA resilience. In this context, we present one paper published in the proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT) and one published in the IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHEs).*

### Contents of this Chapter

---

4.1	SILVER - Statistical Independence and Leakage Verification . . . . .	75
4.2	Transitional Leakage in Theory and Practice – Unveiling Security Flaws in Masked Circuits . . . . .	107

---

## 4.1 SILVER - Statistical Independence and Leakage Verification

### Publication Data

David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - Statistical Independence and Leakage Verification. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020

*This work is reproduced here with permission as stated in the IACR copyright form Part II §6. The author of this doctoral thesis is also an author of this research paper. The copyright of the research paper is held by IACR: ©IACR 2020, 10.1007/978-3-030-64837-4\_26.*

**Content.** In this work, we unify all common security and composability notions in the formal and well-known  $d$ -probing adversary model with respect to the concept of statistical independence between observations made by abstract probes placed on the circuit and the unshared or shared input to the circuit. Leveraging ROBDDs, a common data structure in the context of circuit analysis, we realize an accurate and complete tool for formal verification of SCA resilience: SILVER. SILVER is able to verify a circuit’s security, and its conformity to all common

composability notions (i.e., NI, SNI, and PINI) in the standard, and glitch-extended, robust probing model, as well as the output uniformity of a circuit. We confirm the correct functionality of our methodology and tool through formal arguments as well as by conducting extensive case studies.

**Contribution.** The author was heavily involved in unifying the notions with respect to statistical independence, selecting ROBDDs as a suitable data structure for the actual verification, and deriving a method for performing the actual evaluation leveraging ROBDD. Furthermore, the author was involved in deriving formal arguments for the correctness of the methodology and delivering appropriate case studies. Last but not least, the author significantly contributed to the writing and presentation of the thesis. The author likes to thank both co-authors for their contribution to this work.

# SILVER – Statistical Independence and Leakage Verification

David Knichel<sup>\*[0000-0002-2510-8881]</sup>, Pascal Sasdrich<sup>\*[0000-0002-5443-626X]</sup>, and Amir Moradi<sup>[0000-0002-4032-7433]</sup>

Ruhr University Bochum,  
Horst Görtz Institute for IT Security,  
Bochum, Germany  
{firstname.lastname}@rub.de

**Abstract.** Implementing cryptographic functions securely in the presence of physical adversaries is still a challenge although a lion’s share of research in the physical security domain has been put in development of countermeasures. Among several protection schemes, *masking* has absorbed the most attention of research in both academic and industrial communities, due to its theoretical foundation allowing to provide proofs or model the achieved security level. In return, masking schemes are difficult to implement as the implementation process often is manual, complex, and error-prone. This motivated the need for formal verification tools that allow the designers and engineers to analyze and verify the designs before manufacturing.

In this work, we present a new framework to analyze and verify masked implementations against various security notions using different security models as reference. In particular, our framework – which directly processes the resulting gate-level netlist of a hardware synthesis – particularly relies on Reduced Ordered Binary Decision Diagrams (ROBDDs) and the concept of statistical independence of probability distributions. Compared to existing tools, our framework captivates due to its simplicity, accuracy, and functionality while still having a reasonable efficiency for many applications and common use-cases.

**Keywords:** Verification · Side-Channel Analysis · Probing Security · Reduced Ordered Binary Decision Diagram · Statistical Independence · Probability Distribution.

## 1 Introduction

Even after two decades of research since the seminal description of Side-Channel Analysis (SCA) as a threat to cryptographic implementations [32,33], secure implementation of cryptographically strong algorithms is still a challenging and

---

\* These authors contributed equally to the work.

open problem. In particular, those decades of research have shown that SCA on cryptographic implementations can be performed by observing various physical sources and effects, such as timing [32], power consumption [33], electromagnetic (EM) emanations [27], or temperature and heat dissipation [30]. Eventually, observing the physical characteristics of an electronic device during security-critical cryptographic operations can reveal secret and sensitive information to any observer and adversary. As a consequence, a wide range of protection mechanisms and countermeasures have been proposed to prevent or mitigate any side-channel leakage.

Among all candidates, *masking* (based on the concepts of *secret sharing*) is one of the most promising countermeasures against SCA due to its formal and sound security foundation [18]. As a consequence, many different schemes and variants have been introduced and proposed over the years [31,43,39,42,29,28] to address different implementation and security requirements. Unfortunately, not a few of those schemes have been shown to be insecure due to design flaws or inaccurate models or assumptions [36]. As a result, all these examples confirm that design and implementation of protection mechanisms and countermeasures against SCA is a mostly manual, complex, and error-prone process which requires good understanding of the execution environment and careful consideration of physical and security models.

To this end, an entirely new branch of research started to focus on the development of formal models for adversaries and physical execution environments to simplify and assist in formal verification [31,23,5,26]. Ideally, strong theoretical foundations in security models can assist and help to simplify the design, implementation, and verification of cryptographic implementations and appropriate security mechanisms. In the context of *masking*, formal verification often is conducted in the simple and abstract Ishai-Sahai-Wagner (ISW)  $d$ -probing security model [31] (under some basic assumptions on noise and independence of inputs), which allows an adversary to probe (observe) up to  $d$  intermediate values during the processing of sensitive information.

Due to its conceptual simplicity and level of abstraction, the  $d$ -probing model was rapidly and widely adopted for formal verification [38,6,24,3,4,11,20,44,2]. Indeed, the introduction of this simple but effective security model propelled the automation of formal verification, allowing to reduce the combinatorial complexity of security proofs for masking schemes and their implementations. In fact, development of automated formal verification tools also – in return – stimulated the research and progress on masking schemes, e.g., reducing the cost in terms of randomness [9] or solving the problem of secure composition of masked circuits and gadgets [3,4,17].

However, in its basic manifestation, the  $d$ -probing model does not consider specific physical defaults, such as *glitches*, *transitions*, or *couplings* [26], that may occur during the processing of sensitive information on a physical device. In fact, many schemes proven to be secure in the  $d$ -probing model, eventually fail in security analyzes when concretely implemented. That is mainly due to undesired and unintentional physical defaults that particularly violate the assumption on

the independence of inputs. In particular for hardware implementations, *glitches* are well-known to be an issue and concern for masking schemes [39], wherefore Bloem *et al.* [11] and Faust *et al.* [26] independently proposed an extension of the basic ISW  $d$ -probing model considering glitches for hardware implementations of masking schemes.

In addition, Bloem *et al.* used the concept of Fourier coefficient estimation to implement an automated tool formally verifying the security of masking schemes and their implementations against the basic and glitch-extended  $d$ -probing model. However, due to computational limitations based on the estimation of Fourier coefficients, this tool primarily applies to the security analysis of the first-order setting without consideration of advanced notions such as Non-Interference (NI), Strong Non-Interference (SNI), and Probe-Isolating Non-Interference (PINI). In contrast to this, Barthe *et al.* [2] recently presented a language-based formal verification tool called `maskVerif` which uses the probabilistic information flow to assess the security of masking schemes and their implementations. In particular, using conservative heuristics and an optimistic sampling method, `maskVerif` executes more efficiently than the tool by Bloem *et al.*, while minimizing but still accepting false negatives for non-linear cases.

**Contributions.** In this work, we present and introduce an efficient methodology to analyze and verify the security of masked circuits and implementations under various security notions. Due to a symbolic and exhaustive analysis of probability distributions and statistical independence of joint distributions, we can avoid false negatives and overly conservative decisions. In particular, by means of ROBDDs, a well-known concept and methodology for Integrated Circuit (IC) testing and verification, we formally analyze and verify masked circuits in the ISW  $d$ -probing model even in the presence of *glitches* as physical defaults.

In addition, based on the seminal work of De Meyer *et al.* [21], we reformulate the security notions of  $d$ -probing security,  $d$ -Non-Interference,  $d$ -Strong Non-Interference, and, for the first time,  $d$ -Probe-Isolating Non-Interference based on statistical independence which can be efficiently checked and verified by our tool. Hence, for the first time, state-of-the-art security notions for masked circuits can be analyzed exhaustively without false negatives. Eventually, this contribution is even extended further by efficient verification methods to check and verify the uniformity for output sharings of arbitrary masked circuits.

**Outline.** While Section 2 briefly summarizes our notations and introduces preliminary concepts and notions, including ROBDDs, our circuit model, and security notions, Section 3 is dedicated to a conception and discussion of our verification approach. Besides, Section 3 outlines our leakage models and discusses the main ideas of our verification concept, particularly sketching the application of ROBDDs to check and verify security notions. In Section 4, we then provide formal proofs for all security notions and our leakage verification concept based on statistical independence checks. Before we present details on practical evaluations and experiments in Section 6, we briefly discuss and compare our approach

and concept to essential related work in Section 5. Eventually, we conclude our work in Section 7.

## 2 Background

### 2.1 Notation

We use upper-case characters to denote random variables, bold ones for sets of random variables, and subscripts for elements within a set of variables. Further, let us denote  $\mathbf{X}_{\bar{i}}$  as the set of variables  $\mathbf{X} \setminus X_i$ . Accordingly, we use lower-case characters to denote values a random variable can take, bold ones for sets of values, and subscripts for elements within the set of values. Again, let us denote  $\mathbf{x}_{\bar{i}}$  as the set of values  $\mathbf{x} \setminus x_i$ . In addition, we use  $Pr[X = x]$  for the probability that a random variable  $X$  takes a value  $x$ , while  $Pr[\mathbf{X} = \mathbf{x}]$  denotes the joint probability that each  $X_i \in \mathbf{X}$  takes the value  $x_i \in \mathbf{x}$ . Accordingly, the conditional probability for  $X = x$  given  $Y = y$  is written as  $Pr[X = x|Y = y]$ . Hence,  $Pr[\mathbf{X} = \mathbf{x}|\mathbf{Y} = \mathbf{y}]$  denotes the conditional probability that each  $X_i \in \mathbf{X}$  takes the value  $x_i \in \mathbf{x}$ , if each  $Y_i \in \mathbf{Y}$  takes the value  $y_i \in \mathbf{y}$ . Moreover, the joint distribution over the set  $\mathbf{X}$  is denoted as  $Pr[\mathbf{X}]$ , while  $Pr[\mathbf{X}|\mathbf{Y}] = Pr[\mathbf{X}]$  is simply equivalent to  $Pr[\mathbf{X} = \mathbf{x}|\mathbf{Y} = \mathbf{y}] = Pr[\mathbf{X} = \mathbf{x}]$  for all possible combination of  $\mathbf{x}$  and  $\mathbf{y}$ . Extending this notation,  $Pr[\mathbf{X}|\mathbf{Y}] = Pr[\mathbf{X}|\mathbf{Z}]$  is the same as  $Pr[\mathbf{X} = \mathbf{x}|\mathbf{Y} = \mathbf{y}] = Pr[\mathbf{X} = \mathbf{x}|\mathbf{Z} = \mathbf{z}]$  for all possible combination of  $\mathbf{x}, \mathbf{y}$ , and  $\mathbf{z}$ .

Further, functions are denoted using sans-serif fonts. Handling masked functions, we denote the  $s$ -th share of the a variable as  $X^s$ . Hence, the set of all unshared inputs of a function  $f$  is denoted as  $\mathbf{X} = (X_0, \dots, X_{n-1})$  while the set containing all  $t$  shares of each variable in  $\mathbf{X}$  is denoted as  $Sh(\mathbf{X}) = (X_0^0, X_0^1, \dots, X_0^{t-1}, X_1^0, \dots, X_{n-1}^0, \dots, X_{n-1}^{t-1})$ . Similarly, the set containing all shares of  $X_i \in \mathbf{X}$  is denoted as  $Sh(X_i)$ . Eventually, for a set of indices  $\mathbf{I} \subseteq [0, \dots, t-1]$ ,  $Sh(\mathbf{X})^{\mathbf{I}}$  denotes the set containing all shares  $X_i^s$  with  $0 \leq i < n$  and  $s \in \mathbf{I}$ .

### 2.2 Reduced Ordered Binary Decision Diagrams (ROBDDs)

Binary Decision Diagrams (BDDs) are a basic structure in discrete mathematics and computer science introduced by Akers [1] and refined by Bryant (introducing variable ordering) [15]. In particular, many applications in computer-aided IC design and verification make use of (reduced, ordered) BDDs.

In general, BDDs are concise and unique (i.e., canonical) graph-based representations of Boolean functions  $\mathbb{F}_2^n \rightarrow \mathbb{F}_2$  with a single root node and two terminal nodes (leaves)  $\{\mathbf{T}, \mathbf{F}\}$ . The formal definition of ROBDDs, given in the following paragraphs, is divided into a purely *syntactical* definition, describing the structure based on Directed Acyclic Graphs (DAGs), before providing a *semantical* definition, clarifying the representation of Boolean functions as ROBDDs.



**Syntactical Definition of ROBDDs.** Before providing a syntactical definition for ROBDDs, we first recall the (syntactical) definition of Ordered Binary Decision Diagrams (OBDDs).

**Definition 1 (OBDD Syntax).** An Ordered Binary Decision Diagram is a pair  $(\pi, \mathcal{G})$ , where  $\pi$  denotes the variable ordering of the OBDD and  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  is a finite DAG with vertices  $\mathcal{V}$ , edges  $\mathcal{E}$ , and the following properties:

- (1) There is a single root node and each node  $v \in \mathcal{V}$  is either a non-terminal node or one of two terminal nodes  $\{\mathbf{T}, \mathbf{F}\}$ .
- (2) Each non-terminal node  $v$  is labeled with a variable in  $\mathbf{X}$ , with  $|\mathbf{X}| = n$ , denoted as  $\text{var}(v)$ , and has exactly two child nodes in  $\mathcal{V}$  which are denoted as  $\text{then}(v)$  and  $\text{else}(v)$ .
- (3) For each path from the root node to a terminal node, the variables in  $\mathbf{X}$  are encountered at most once and in the same order defined by the variable ordering  $\pi$ . More precisely, the variable ordering  $\pi$  of an OBDD is a bijection  $\pi : \{1, 2, \dots, n\} \rightarrow \mathbf{X}$ .

Furthermore, assuming the following two restrictions ensures a concise and canonical representation (under a given variable ordering  $\pi$ ), defined as ROBDD.

**Definition 2 (ROBDD Syntax).** An OBDD is called Reduced Ordered Binary Decision Diagram, if and only if there is no node  $v \in \mathcal{V}$  such that  $\text{then}(v) = \text{else}(v)$  and there are no duplicate nodes, i.e., two nodes  $\{v, v'\} \in \mathcal{V}$  such that  $\text{var}(v) = \text{var}(v')$ ,  $\text{then}(v) = \text{then}(v')$ , and  $\text{else}(v) = \text{else}(v')$ .

**Semantical Definition of ROBDDs.** Each ROBDD with root  $v \in \mathcal{V}$  recursively defines a Boolean function  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$  according to the following definition.

**Definition 3 (ROBDD Semantics).** An ROBDD over  $\mathbf{X}$  represents a Boolean function  $f$  recursively carried out at each node and defined as follows:

- (1) If  $v$  is the terminal node  $\mathbf{F}$ , then  $f_v|_x = 0$ , otherwise, if  $v$  is the terminal node  $\mathbf{T}$ , then  $f_v|_x = 1$ .
- (2) If  $v$  is a non-terminal node and  $\text{var}(v) = x_i$ , then  $f_v$  is defined by the Shannon decomposition  $f_v = x_i \cdot f_{\text{then}(v)|_{x_i=1}} + \bar{x}_i \cdot f_{\text{else}(v)|_{x_i=0}}$ .

**Boolean Operations over ROBDDs.** Given the syntactical and semantical definitions for ROBDDs, we now can define arbitrary Boolean operations over Boolean functions  $f_{v_1}$  and  $f_{v_2}$  represented by two ROBDDs with root nodes  $v_1$  and  $v_2$ . In particular, let  $f = f_{v_1} \circ f_{v_2}$  where  $\circ$  denotes any binary Boolean operation, then the ROBDD for  $f$  can be derived and composed recursively as:

$$\begin{aligned}
 f &= x_i \cdot f|_{x_i=1} + \bar{x}_i \cdot f|_{x_i=0} \\
 &= x_i \cdot (f_{v_1} \circ f_{v_2})|_{x_i=1} + \bar{x}_i \cdot (f_{v_1} \circ f_{v_2})|_{x_i=0} \\
 &= x_i \cdot (f_{v_1}|_{x_i=1} \circ f_{v_2}|_{x_i=1}) + \bar{x}_i \cdot (f_{v_1}|_{x_i=0} \circ f_{v_2}|_{x_i=0})
 \end{aligned} \tag{1}$$

### 2.3 Circuit Model

For the remainder of this work, we consider and model a deterministic circuit  $C$  as a DAG, where the vertices are combinational gates and edges are wires carrying elements from the finite field  $\mathbb{F}_2$ .

**Physical Model.** Without loss of generality, the physical structure of a deterministic circuit  $C$  at gate level is modeled using the set of combinational gates  $\{\text{not}, \text{and}, \text{nand}, \text{or}, \text{nor}, \text{xor}, \text{xnor}\}$  (with fan-in at most 2) while all sequential memory gates `reg` model a clock-dependent synchronization point. Further, each Boolean input variable is associated with a single in gate (with fan-in 0), while the output and result of a Boolean function is associated with a single out gate (with fan-out 0). Eventually, `ref` are special-purpose gates with fan-in 0 that introduce a *independently and identically distributed* (i.i.d) random element from the finite field  $\mathbb{F}_2$ .

**Functional Model.** Each deterministic circuit  $C$  realizes an  $n \times m$  vectorial Boolean function  $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$  given its *coordinate functions*  $f_1, \dots, f_m$  defined over  $\mathbf{X} \in \mathbb{F}_2^n$ . In particular, each  $X_i \in \mathbf{X}$  is assumed to be *independently and identically distributed* (i.i.d) and associated with a single in gate, while each  $f_i$  is associated with a single out gate.

Further, the function of each gate within the circuit model  $C$  is derived recursively over the functions of its fan-in gates by means of Boolean operations over ROBDDs. Hence, each gate in the circuit model itself can be considered as a Boolean function over (a subset of) the inputs  $\mathbf{X} \in \mathbb{F}_2^n$  and we can introduce a functional abstraction layer to the physical circuit model using ROBDDs to canonically represent and store the derived Boolean functions.

**Security Model.** Eventually, security critical circuits handling a sensitive secret  $X$  are associated with a security order  $d$  and protected (masked) based on Boolean sharing. This means, each security critical and sensitive secret  $X$  is shared with at least  $d + 1$  shares such that  $X = \bigoplus_0^d X^i$ . Similarly, sensitive and security critical outputs of a masked circuit are shared using Boolean sharing, such that  $F(\mathbf{X}) = \bigoplus_0^d F^i(\mathbf{X})$ .

### 2.4 Security Notions

Before introducing our verification approach and methodology to analyze an arbitrary circuit under various security notions, we first introduce the definitions of all necessary security notions. In particular, our security definitions are based on the work in [21] while we reformulate the definitions in order to provide generalizations from circuits with  $d + 1$  input shares to circuits with an arbitrary number of shares when examining  $d$ th-order security. In addition, we extend the definitions from single sensitive and secret variable to a set of arbitrary number of secret variables.

**Probing Security.** Probing security is defined as the probes being statistically independent of any sensitive input. More precisely, the joint distribution of the considered set of probes has to be independent of the joint distribution of all sensitive inputs. This can be formally defined as:

**Definition 4 (*d*-Probing Security.).** A circuit  $C$  with secret input set  $\mathbf{X} \in \mathbb{F}_2^n$  is *d*-probing secure, if and only if for any observation set  $\mathbf{Q}$  containing *d* wires,  $\mathbf{X}$  is statistically independent of the observation set, i.e., the following condition holds:

$$\Pr[\mathbf{Q}|\mathbf{X}] = \Pr[\mathbf{Q}] \quad (2)$$

**Non-Interference.** The notion of *Non-Interference* allows partial information on the sensitive inputs becoming available to the adversary through probing the circuit. In particular, if the observed circuit is *d*-NI, the adversary is not able to successfully distinguish the circuit result from a simulator working on partial information knowing, i.e., using at most *d* shares of each input.

More formally, each adversarial probe set should be perfectly simulatable knowing only a subset of all shares of each input. Let  $\mathbf{S}$  be a set over arbitrary input shares  $X_i^j$ , i.e.,  $\mathbf{S} \subset Sh(\mathbf{X})$ , and  $|\mathbf{S}|_i$  denote the number of shares in  $\mathbf{S}$  that correspond to input  $X_i$ . In order to guarantee *d*-NI, there exist a simulation set  $\mathbf{S}$  with  $|\mathbf{S}|_{\forall i} \leq d$  for which a probe in  $\mathbf{Q}$  is perfectly simulatable, i.e., an attacker is not able to distinguish between a simulation of  $C$  using only elements in  $\mathbf{S}$  from the observations of  $C$  even when knowing all input shares. This can be directly translated into the condition that there has to exist a simulation set  $\mathbf{S}$  with  $|\mathbf{S}|_{\forall i} \leq d$ , for which the distributions  $\Pr[\mathbf{Q}|\mathbf{S}]$  and  $\Pr[\mathbf{Q}|Sh(\mathbf{X})]$  are equal.

**Definition 5 (*d*-Non-Interference).** A circuit  $C$  with secret input set  $\mathbf{X} \in \mathbb{F}_2^n$  provides *d*-Non-Interference if and only if for any observation set of  $t \leq d$  wires  $\mathbf{Q}$  there exists a set  $\mathbf{S}$  of input shares with  $|\mathbf{S}|_{\forall i} \leq t$  such that

$$\Pr[\mathbf{Q}|\mathbf{S}] = \Pr[\mathbf{Q}|Sh(\mathbf{X})]. \quad (3)$$

**Strong Non-Interference.** The notion of SNI has been introduced as extension to NI correcting deficiencies in terms of composability of secure gadgets within a circuit. In contrast to NI, any probe on a circuit output (also, through composition, considered as input to a subsequent gadget) is not allowed to give information about any share in the input. This means, each probe on an output wire must be perfectly simulatable without knowledge of any input shares in order to stop the flow of sensitive information between composed gadgets.

More formally, each adversarial probe set again should be perfectly simulatable knowing only a subset of all shares of each input. However, for a set  $\mathbf{Q}$  of *d* probes with  $t_1$  probes on internal wires and  $t_2$  probes on output wires while  $t_1 + t_2 \leq d$ , the size of the set  $\mathbf{S}$  is bounded by the internal probes only, i.e.,  $|\mathbf{S}|_{\forall i} \leq t_1$ . This directly translates into the following definition and condition.

**Definition 6 (*d*-Strong Non-Interference).** A circuit  $C$  with secret input set  $\mathbf{X} \in \mathbb{F}_2^n$  provides *d*-Strong Non-Interference if and only if for any observation set of  $t = t_1 + t_2 \leq d$  wires  $\mathbf{Q}$  of which  $t_1$  are internal wires and  $t_2$  are output wires, there exists a simulation set  $\mathbf{S}$  of input shares with  $|\mathbf{S}|_{\forall i} \leq t_1$  such that Equation (3) holds.

**Probe-Isolating Non-Interference.** Unfortunately, the security notion of SNI in practice is often very conservative and inefficient as it introduces more area and randomness than necessary to achieve certain security goals. To address this issue, Cassiers *et al.* introduced the notion of Probe-Isolating Non-Interference [17] which offers trivial composition of any gadgets inspired by the trivial composition of linear functions and the concept of sharing domain separations as introduced in [29]. As the original SNI definition limits composability to single-output gadgets, Cassiers *et al.* also introduced a generalization of SNI to gadgets with multiple inputs and multiple outputs (Multiple-Input-Multiple-Output SNI (MIMO-SNI)). This is a very strong notion which in fact already implies security under the PINI notion, i.e., every gadget that provides *d*-MIMO-SNI also provides *d*-PINI. As PINI already guarantees trivial composition, we do not consider MIMO-SNI any further in this work.

In the context of PINI, each circuit input and output is assigned a unique circuit share index (i.e., a share domain) and any probe set on these wires should be perfectly simulatable knowing only the set of inputs that are assigned to the same circuit share index. Further, any additional probe on internal wires gives the adversary access to at most one additional circuit share, i.e., must be perfectly simulatable knowing only the according set of inputs assigned to these circuit shares. Eventually, this translates to the following definition.

**Definition 7 (*d*-Probe-Isolating Non-Interference).** Let  $\mathbf{P}$  be the set of internal probes with  $|\mathbf{P}| = t_1$ . Let further  $\mathbf{I}_O$  be the index set assigned to the probed output wires  $\mathbf{O}$  with  $|\mathbf{I}_O| = t_2$ .

A circuit  $C$  with secret input set  $\mathbf{X} \in \mathbb{F}_2^n$  provides *d*-Probe-Isolating Non-Interference if and only if for every  $\mathbf{P}$  and  $\mathbf{O}$  with  $t_1 + t_2 \leq d$  there exists a set  $\mathbf{I}_I$  of circuit indices with  $|\mathbf{I}_I| \leq t_1$  such that  $\mathbf{Q} = \mathbf{P} \cup \mathbf{O}$  can be perfectly simulated by  $\mathbf{S} = \text{Sh}(\mathbf{X})_{\mathbf{I}_I \cup \mathbf{I}_O}$ , i.e., Equation (3) holds.

**Uniformity.** The security of (Boolean) masking schemes relies on a fundamental assumption: *uniform sharing*. For that, the initial sharing of any secret variable  $X$  using  $d + 1$  shares, such that  $X = \bigoplus_0^d X^i$ , can be done by assigning random values to  $X^0, \dots, X^{d-1}$  and deriving  $X^d = X \oplus \bigoplus_0^{d-1} X^i$ . Such a sharing then is uniform if all random variables  $X^0, \dots, X^{d-1}$  are independent of each other and have a uniform distribution over  $\mathbb{F}_2$ .

In practice, the uniformity of the output sharing of gadgets has been defined as a fundamental requirement for Threshold Implementations (TIs), particularly for secure composition of gadgets [39]. Otherwise, a non-uniform output sharing of a gadget becomes the non-uniform input sharing of another gadget, hence

violating the essential assumption of uniformity for (Boolean) masking schemes. Note, however, that a gadget can be probing secure, but it is not necessarily uniform. Likewise, a uniform gadget does not automatically lead to a probing-secure construction. This has been handled specifically in NI and SNI gadgets, e.g., by injecting fresh randomness at the output thereby refreshing the output sharing, i.e., achieving uniformity.

Definition 8 formalizes the notion of a uniform sharing as it states that for every unshared value, each valid sharing has to occur with the same probability.

**Definition 8 (Uniform Sharing).** *Let  $\mathbf{Y}$  be a set of binary random variable and  $Sh(\mathbf{Y})$  its corresponding Boolean sharing. Then  $Sh(\mathbf{Y})$  is said to be a uniform sharing iff for some constant  $p$*

$$Pr[Sh(\mathbf{Y}) = \mathbf{y}^* | \mathbf{Y} = \mathbf{y}] = \begin{cases} p & \text{if } \mathbf{y}^* \text{ is a valid sharing for } \mathbf{y} \\ 0 & \text{else} \end{cases} . \quad (4)$$

### 3 Verification Concept

This section briefly elaborates our main idea and concept for our verification model and approach.

#### 3.1 Leakage Models

For verification, we additionally consider each security notion under two different leakage models denoted *standard* respectively *robust* leakage model.

**Standard Leakage Model.** For our standard leakage model following the concept of the traditional ISW  $d$ -probing model [31], we assume an ideal circuit without any physical defaults such as *glitches* or *transitions*. In practice, this leakage model relates to a software scenario where each result of an operation (i.e., a gate in a circuit  $\mathbf{C}$ ) is stored in a register before it is used by subsequent operations (gates). Note that in this model, the implementation platform's specific effects like pipelines are entirely ignored. In this model, an adversarial probe provides access to the field element carried on the probed wire. More precisely, the adversary gains full access to the Boolean function represented by the driving gate in order to derive the field element.

**Robust Leakage Model.** In contrast to our standard leakage model, for our robust model following the leakage model in [26], we also take physical default in terms of glitches into consideration, hence, in practice this model relates to a hardware scenario. Since only circuit inputs and memory elements are assumed to switch synchronous to a circuit clock, glitches will propagate through all combinational gates between two synchronization points. Therefore, by probing a wire, the adversary not only gains access to the field element of the driving gate

but also can access all stable field elements of the last synchronization points which drive the probed gate (having a path to the driving gate in the circuit graph). More precisely, the adversary gains full access to the set of these field elements (and any subset) through so called *glitch-extended* probes.

### 3.2 Verification Approach

Based on some fundamental observation, this section outlines our basic concept and explains our main approach to verify different security notions starting from a circuit model given as gate-level netlist.

**Random Variables with Binary Events.** According to our circuit model, each edge in the circuit graph models a wire and carries an element from the field  $\mathbb{F}_2$  with two elements. Thus, we first observe each wire, and its associated field element can be modeled as a binary random variable defined over the sample space  $\Omega = \{0,1\}$  of two basic events given the assumption that all primary circuit inputs are *independent and identically distributed* (i.i.d.). Based on this observation, we can use the probability distributions of all random variables in order to analyze and verify a circuit model against the security notions defined in Section 2.4.

**Probability Distribution and Satisfiability.** In general, the probability of an event is defined by the sum of the probabilities of all *outcomes* that satisfy the event. In the context of our circuit model, an *outcome* can be considered as a variable assignment to the primary circuit inputs that leads to the desired element of the sample space at the observed random variable. For this, computing the probability density function of a random variable associated with a circuit wire reduces to enumerating and counting the primary input variable assignments that satisfy the corresponding basic events for the observed random variable.

**Symbolic Simulation using ROBDDs.** As the naive approach of exhaustive and literal simulation of the circuit model expeditiously becomes infeasible with increasing circuit complexity and number of primary circuit inputs, symbolic simulation and analysis is necessary to maintain the generation of all probability distributions practicable even for large and complex circuit models. More precisely, each gate in the circuit model represents a sub-circuit and is associated with a Boolean function given as ROBDD that computes the gate output over the set of primary circuit inputs. Since ROBDDs are concise and canonical representations of Boolean functions, counting the number of *cubes*, i.e., the satisfying variable assignments, for each basic binary event can be done efficiently using symbolic analysis. Knowing the total number of possible variable assignments, computing the probability distribution for each random variable remains feasible even for large and complex circuits.

**Standard and Glitch-Extended Probes.** Considering our two different leakage models, we also have to differentiate the capabilities and knowledge of the adversary. Firstly, for the standard model we thus assume that an adversarial probe gives access to the probability distribution of a field element carried on an arbitrary wire observed by the adversarial probe. More precisely, the adversary in this case learns the Boolean sub-function associated with the driving gate in order to compute the field element and its probability distribution as function of the primary circuit inputs. Secondly, in contrast to the standard model, a robust or *glitch-extended* probe extends the capabilities and knowledge of the adversary as it also provides access to the joint distribution of all hindmost contributing synchronization points (memory elements or primary inputs). Hence, in order to model physical defaults and in particular glitches, the adversary also learns the Boolean sub-functions associated with the corresponding synchronization elements.

**Statistical Independence and Security Checks.** Eventually, depending on the targeted security order  $d$ , an adversarial *observation* can consist of up to  $d$  independently placed adversarial probes and the adversary is allowed to combine the information and knowledge of all probes to learn details of the secret. In order to verify security under the given security notions as defined in Section 2.4, we perform an exhaustive exploration and check of all possible adversarial observations  $\mathbf{Q}$  combining up to  $d$  probes. For this, the following section is dedicated to a detail description and verification of our performed security checks.

## 4 Statistical Independence and Security Checks

Before formally analysis and verification of the correctness of our security checks, we briefly recap the definitions of (joint) probability mass functions and statistical independence for sets of random variables.

### 4.1 Statistical Independence

The probability mass function provides the probability of all possible values for a set of discrete random variables based on their probability distribution.

**Definition 9 (Probability Mass Function).** *Let  $\mathbf{X}$  be a set of discrete random variables. The probability mass function  $p_{\mathbf{X}}(\mathbf{x})$  is defined as:*

$$p_{\mathbf{X}}(\mathbf{x}) = Pr[\mathbf{X} = \mathbf{x}].$$

Based on this, given two arbitrary sets of discrete random variables, the joint probability mass function between these two variable sets then is defined as follows.

**Definition 10 (Joint Probability Mass Function).** Let  $\mathbf{X}, \mathbf{Y}$  be two sets of discrete random variables. The joint probability mass function  $p_{\mathbf{X}, \mathbf{Y}}(\mathbf{x}, \mathbf{y})$  is defined as:

$$p_{\mathbf{X}, \mathbf{Y}}(\mathbf{x}, \mathbf{y}) = \Pr[\mathbf{X} = \mathbf{x} \text{ and } \mathbf{Y} = \mathbf{y}].$$

Using the definitions of probability mass function and joint probability mass function, we can express the notion of statistical independence of two sets of discrete random variables according to the following definition.

**Definition 11 (Statistical Independence).** Let  $\mathbf{X}, \mathbf{Y}$  be two sets of discrete random variables.  $\mathbf{X}, \mathbf{Y}$  are statistically independent if and only if the joint probability mass function for  $\forall \mathbf{x}$  and  $\forall \mathbf{y}$  satisfies

$$p_{\mathbf{X}, \mathbf{Y}}(\mathbf{x}, \mathbf{y}) = p_{\mathbf{X}}(\mathbf{x}) \cdot p_{\mathbf{Y}}(\mathbf{y}).$$

**Statistical Independence of Binary Random Variables.** In the context of our security notions, we are mainly interested in statistical independence of *binary random variables*. As any binary random variable can only take two different events, Theorem 1 states that checking statistical independence for one event implies statistical independence for both events, even extending to the case of sets of binary random variables.

**Theorem 1.** Let  $\mathbf{X}, \mathbf{Y}$  be two sets of binary random variables. Then  $\mathbf{X}$  and  $\mathbf{Y}$  are statistically independent, if and only if  $p_{\mathbf{X}', \mathbf{Y}'}(\mathbf{a}, \mathbf{b}) = p_{\mathbf{X}'}(\mathbf{a}) \cdot p_{\mathbf{Y}'}(\mathbf{b})$  for any fixed values  $\mathbf{a}$  and  $\mathbf{b}$  and every possible combination of  $\mathbf{X}' \subseteq \mathbf{X}$  and  $\mathbf{Y}' \subseteq \mathbf{Y}$ .

In order to proof correctness of Theorem 1, we start with the basic case of two binary random variables (i.e., sets of cardinality one).

**Lemma 1.** Let  $X, Y \in \mathbb{F}_2$  be two binary random variables. Then, a necessary and sufficient condition for  $X$  to be statistically independent of  $Y$  is that, for any fixed values  $a, b \in \{0, 1\}$ , it holds

$$p_{X, Y}(a, b) = p_X(a) \cdot p_Y(b).$$

*Proof.* According to Definition 11, the necessity of this proposition is obvious, hence, the proof focuses on the sufficiency. Without loss of generality, we now assume Lemma 1 is true for  $a = b = 1$ , i.e.,  $p_{X, Y}(1, 1) = p_X(1) \cdot p_Y(1)$ . Since  $X = 0$  and  $X = 1$  are counter events for binary variables, both the fact  $p_X(0) + p_X(1) = 1$  and the fact  $p_{X, Y}(0, 1) + p_{X, Y}(1, 1) = p_Y(1)$  hold, and we have

$$\begin{aligned} p_{X, Y}(0, 1) + p_{X, Y}(1, 1) &= p_Y(1) \\ \Leftrightarrow p_{X, Y}(0, 1) + p_X(1) \cdot p_Y(1) &= p_Y(1) \\ \Leftrightarrow p_{X, Y}(0, 1) &= (1 - p_X(1)) \cdot p_Y(1) \\ \Leftrightarrow p_{X, Y}(0, 1) &= p_X(0) \cdot p_Y(1) \end{aligned}$$

Proving the cases for  $a = 1, b = 0$  and  $a = b = 0$  is trivial as it follows the same approach, hence is left out for brevity.  $\square$



In a next step, we extend the basic case through mathematical induction in order to prove statistical independence between a single random binary variable and a set of random binary variables.

**Lemma 2.** *Let  $X$  be a binary random variable and  $\mathbf{Y}$  a set of  $n$  random binary variables. Further, let  $X$  be statistically independent of the joint distribution of  $\mathbf{Y}$ . Now, the joint distribution  $\mathbf{Y}^+$ , with  $\mathbf{Y} \subset \mathbf{Y}^+$  and  $|\mathbf{Y}^+| = n+1$  is statistically independent of  $X$ , if and only if  $p_{X, \mathbf{Y}^+}(a, \mathbf{b}^+) = p_X(a) \cdot p_{\mathbf{Y}^+}(\mathbf{b}^+)$  for any fixed values  $a, \mathbf{b}^+$ .*

*Proof.* We now give a formal proof using mathematical induction on  $n$ .

Base case: We first show that Lemma 2 holds for  $n = 0$ .

Clearly, if  $n = 0$ ,  $\mathbf{Y}$  is the empty set while  $\mathbf{Y}^+$  is a single binary random variable. Then, according to Lemma 1,  $X$  and  $\mathbf{Y}^+$  are statistically independent if and only if for any fixed values  $a, b$  it holds that  $p_{X, \mathbf{Y}^+}(a, b) = p_X(a) \cdot p_{\mathbf{Y}^+}(b)$ .  $\square$

Induction: If Lemma 2 holds for  $n = k$ , it also holds for  $n = k + 1$  with  $k \geq 0$ .

For this, we first show that, without loss of generality, for  $X, \mathbf{Y}^+$  the following fact  $p_{X, \mathbf{Y}^+}(a, \mathbf{b}_i^+) = p_X(a) \cdot p_{\mathbf{Y}^+}(\mathbf{b}_i^+)$  with  $\mathbf{b}_i^+ = \{y_0, y_1, \dots, \bar{y}_i, \dots, y_k, y_{k+1}\}$  holds, if:

- (i)  $p_{X, \mathbf{Y}}(a, \mathbf{b}) = p_X(a) \cdot p_{\mathbf{Y}}(\mathbf{b})$  with  $\mathbf{b} = \{y_0, y_1, \dots, y_{i-1}, y_{i+1}, \dots, y_k, y_{k+1}\}$ ,
- (ii)  $p_{X, \mathbf{Y}^+}(a, \mathbf{b}^+) = p_X(a) \cdot p_{\mathbf{Y}^+}(\mathbf{b}^+)$  with  $\mathbf{b}^+ = \{y_0, y_1, \dots, y_i, \dots, y_k, y_{k+1}\}$

Further, we note that for binary random variables it always holds that:

$$p_{X, \mathbf{Y}}(a, \mathbf{b}) = p_{X, \mathbf{Y}^+}(a, \mathbf{b}^+) + p_{X, \mathbf{Y}^+}(a, \mathbf{b}_i^+)$$

Given that (i), (ii) are conditions for Lemma 2, we can state the following:

$$\begin{aligned} p_{X, \mathbf{Y}}(a, \mathbf{b}) &= p_X(a) \cdot p_{\mathbf{Y}}(\mathbf{b}) \\ \Leftrightarrow p_{X, \mathbf{Y}^+}(a, \mathbf{b}^+) + p_{X, \mathbf{Y}^+}(a, \mathbf{b}_i^+) &= p_X(a) \cdot p_{\mathbf{Y}}(\mathbf{b}) \\ \Leftrightarrow p_X(a) \cdot p_{\mathbf{Y}^+}(\mathbf{b}^+) + p_{X, \mathbf{Y}^+}(a, \mathbf{b}_i^+) &= p_X(a) \cdot p_{\mathbf{Y}}(\mathbf{b}) \\ \Leftrightarrow p_{X, \mathbf{Y}^+}(a, \mathbf{b}_i^+) &= p_X(a) \cdot p_{\mathbf{Y}}(\mathbf{b}) - p_X(a) \cdot p_{\mathbf{Y}^+}(\mathbf{b}^+) \\ \Leftrightarrow p_{X, \mathbf{Y}^+}(a, \mathbf{b}_i^+) &= p_X(a) \cdot [p_{\mathbf{Y}}(\mathbf{b}) - p_{\mathbf{Y}^+}(\mathbf{b}^+)] \\ \Leftrightarrow p_{X, \mathbf{Y}^+}(a, \mathbf{b}_i^+) &= p_X(a) \cdot p_{\mathbf{Y}^+}(\mathbf{b}_i^+) \end{aligned}$$

As the sorting of variables in  $\mathbf{Y}^+$  is not fixed, this approach also extends to inversion of any other event and therefore can easily be extended to show statistical independence for every combination of events.  $\square$

Eventually, this also proves Theorem 1. In particular, knowing that  $\mathbf{X}, \mathbf{Y}$  are statistically independent, we can argue that  $\mathbf{X}^+, \mathbf{Y}$  with  $\mathbf{X} \subset \mathbf{X}^+$ ,  $|\mathbf{X}| = n$ , and  $|\mathbf{X}^+| = n + 1$  must be statistically independent, if and only if  $p_{\mathbf{X}^+, \mathbf{Y}}(\mathbf{a}^+, \mathbf{b}) = p_{\mathbf{X}^+}(\mathbf{a}^+) \cdot p_{\mathbf{Y}}(\mathbf{b})$  using the same approach as for Lemma 2.

---

**Algorithm 1:** Explore  $d$ -Probing Security.

---

**Input** :  $\mathbf{X}$  – Set of  $n$  secret variables.  
**Output**:  $\mathbf{Q}$  – Set of  $d + 1$  successful probes.

```

1  $d \leftarrow 1$ 
2 while true do
3   foreach probing set  $\mathbf{Q}$  with  $|\mathbf{Q}| = d$  do
4     for  $t = 1$  to  $n$  do
5       foreach  $\mathbf{X}' \subseteq \mathbf{X}$  with  $|\mathbf{X}'| = t$  do
6         if  $p_{\mathbf{Q}, \mathbf{X}'}(\mathbf{1}, \mathbf{1}) \neq p_{\mathbf{Q}}(\mathbf{1}) \cdot p_{\mathbf{X}'}(\mathbf{1})$  then
7           return  $\mathbf{Q}$ 
8         end
9       end
10    end
11  end
12   $d \leftarrow d + 1$ 
13 end

```

---

## 4.2 $d$ -Probing Security

Checking  $d$ -probing security according to Definition 4 requires to verify statistical independence of the set of secret variables and any observation of at most  $d$  probes. This section presents an exploration algorithm that allows to find and verify the maximum security order of a given circuit with secret variables  $\mathbf{X}$ . Eventually, the algorithm will return the first set of  $d + 1$  probes that is not statistically independent of the secret variables.

**Algorithmic Verification Approach.** Algorithm 1 presents our algorithmic approach to explore and verify  $d$ -probing security of a Circuit Under Test (CUT). In general, the algorithm is initialized with  $d = 1$ , i.e., starts to explore and verify first-order security before extending verification to higher orders. Since for  $|\mathbf{Q}| = 1$  each observation set contains only a single binary variable (observed by a single probe placed on a wire within the circuit  $\mathbf{C}$ ), according to Theorem 1 it is sufficient to verify:

$$p_{\mathbf{Q}, \mathbf{X}'}(\mathbf{1}, \mathbf{1}) \stackrel{?}{=} p_{\mathbf{Q}}(\mathbf{1}) \cdot p_{\mathbf{X}'}(\mathbf{1}) \quad (5)$$

for all possible combinations of secret variables  $\mathbf{X}' \subseteq \mathbf{X}$ . If any of those checks fails, the current observation is not statistically independent of the secret variables and Algorithm 1 terminates with returning the current set of observation indicating the security of the CUT to be at most  $d = |\mathbf{Q}| - 1$ .

If probing security is verified for all joint distributions of  $d$  probes, the algorithm continues with all combinations of  $d + 1$  probes. However, for independence of the current set of probes  $\mathbf{Q}$ , it is still sufficient to check Equation (5) for all combinations of secret variables (but only the current combination of probes), since any subset of probes has already been verified during previous iterations (for smaller  $d$ ).

---

**Algorithm 2:** Explore  $d$ -Non-Interference Security.
 

---

**Input** :  $Sh(\mathbf{X})$  – Set of all shares of  $n$  secret variables  $\mathbf{X}$ .  
**Output:**  $\mathbf{Q}$  – Set of  $d + 1$  successful probes.

```

1  $d \leftarrow 1$ 
2 while true do
3   foreach probing set  $\mathbf{Q}$  with  $|\mathbf{Q}| = d$  do
4     simulatable  $\leftarrow$  true
5     for  $t = 0$  to  $d$  do
6       foreach  $\mathbf{S} \subseteq Sh(\mathbf{X})$  with  $|\mathbf{S}|_{\forall i} = t$  do
7         if  $p_{\mathbf{Q}, Sh(\mathbf{X})}(\mathbf{1}, \mathbf{1}) \neq p_{\mathbf{Q}, \mathbf{S}}(\mathbf{1}, \mathbf{1}) \cdot p_{\bar{\mathbf{S}}}(\mathbf{1})$  then
8           simulatable  $\leftarrow$  false
9         end
10      end
11    end
12    if not simulatable then
13      return  $\mathbf{Q}$ 
14    end
15  end
16   $d \leftarrow d + 1$ 
17 end
  
```

---

Eventually, all verification and statistical independence checks are performed based on ROBDDs in order to generate all (joint) probability mass functions  $p_{\mathbf{Q}}$ ,  $p_{\mathbf{X}}$ , and  $p_{\mathbf{Q}, \mathbf{X}}$ . In particular, evaluation of the probability mass functions for  $\mathbf{1}$  is very efficient for ROBDD-based representations, usually implemented as *satisfiability-check*.

### 4.3 $d$ -Non-Interference

Checking  $d$ -NI security using Definition 5 requires to verify Equation (3) that every set of at most  $d$  probes  $\mathbf{Q}$  on a circuit  $\mathbf{C}$  has to be perfectly simulatable using only a subset  $\mathbf{S}$  of all shares of the secret variables  $\mathbf{X}$ . Using the concept of statistical independence of two sets of random binary variables, we can express NI using the following theorem.

**Theorem 2.** *Let  $\bar{\mathbf{S}} := Sh(\mathbf{X}) \setminus \mathbf{S}$ . Since all input shares are i.i.d., Equation (3) simplifies to:*

$$p_{\mathbf{Q}, Sh(\mathbf{X})}(\mathbf{q}, \mathbf{x}) = p_{\mathbf{Q}, \mathbf{S}}(\mathbf{q}, \mathbf{s}) \cdot p_{\bar{\mathbf{S}}}(\bar{\mathbf{s}}). \quad (6)$$

*In particular, since  $Sh(\mathbf{X}) = \mathbf{S} \cup \bar{\mathbf{S}}$ , we can simply verify statistical independence of  $\mathbf{Q} \cup \mathbf{S}$  and  $\bar{\mathbf{S}}$  (with  $\mathbf{x} = \mathbf{s} \cup \bar{\mathbf{s}}$ ).*

---

**Algorithm 3:** Explore  $d$ -Strong Non-Interference Security.

---

**Input** :  $Sh(\mathbf{X})$  – Set of all shares of  $n$  secret variables  $\mathbf{X}$ .  
**Output:**  $\mathbf{Q}$  – Set of  $d + 1$  successful probes.

```

1  $d \leftarrow 1$ 
2 while true do
3   foreach probing set  $\mathbf{Q}$  with  $|\mathbf{Q}| = t_1 + t_2 \leq d$  ( $t_1$  internal,  $t_2$  output
   probes) do
4     simulatable  $\leftarrow$  true
5     for  $t = 0$  to  $t_1$  do
6       foreach  $\mathbf{S} \subseteq Sh(\mathbf{X})$  with  $|\mathbf{S}|_{v_i} = t$  do
7         if  $p_{\mathbf{Q}, Sh(\mathbf{X})}(\mathbf{1}, \mathbf{1}) \neq p_{\mathbf{Q}, \mathbf{S}}(\mathbf{1}, \mathbf{1}) \cdot p_{\bar{\mathbf{S}}}(\mathbf{1})$  then
8           simulatable  $\leftarrow$  false
9         end
10      end
11    end
12    if not simulatable then
13      return  $\mathbf{Q}$ 
14    end
15  end
16   $d \leftarrow d + 1$ 
17 end

```

---

*Proof.*

$$\begin{aligned}
Pr[\mathbf{Q}|\mathbf{S}] &= Pr[\mathbf{Q}|Sh(\mathbf{X})] \\
&\Leftrightarrow Pr[\mathbf{Q}, \mathbf{S}] \cdot Pr[Sh(\mathbf{X})] = Pr[\mathbf{Q}, Sh(\mathbf{X})] \cdot Pr[\mathbf{S}] \\
&\stackrel{\text{i.i.d., } Sh(\mathbf{X})}{\Leftrightarrow} Pr[\mathbf{Q}, \mathbf{S}] \cdot Pr[\bar{\mathbf{S}}] = Pr[\mathbf{Q}, Sh(\mathbf{X})] \\
&\Leftrightarrow p_{\mathbf{Q}, \mathbf{S}}(\mathbf{q}, \mathbf{s}) \cdot p_{\bar{\mathbf{S}}}(\bar{\mathbf{s}}) = p_{\mathbf{Q}, Sh(\mathbf{X})}(\mathbf{q}, \mathbf{x})
\end{aligned}$$

□

**Algorithmic Verification Approach.** Algorithm 2 explores and verifies  $d$ -NI for increasing  $d$  and all possible observations  $\mathbf{Q}$  of at most  $d$  probes. In particular, the algorithm proceeds as soon as a successful simulation set  $\mathbf{S}$  of input shares is found for the current set of probes  $\mathbf{Q}$ , such that  $\mathbf{Q}$  is perfectly simulatable using  $\mathbf{S}$ . However, if the algorithm encounters a set of probes  $\mathbf{Q}$  with  $|\mathbf{Q}| = d + 1$  which is not simulatable for set of input shares (according to the definition of NI), the algorithm terminates and returns the current set of probes indicating  $d$ -NI with  $d = |\mathbf{Q}| - 1$ .

#### 4.4 $d$ -Strong Non-Interference

Checking  $d$ -SNI is very similar to checking  $d$ -NI, except for stronger constraints on the simulation set  $\mathbf{S}$  due to stronger distinction between internal and output probes.

---

**Algorithm 4:** Explore  $d$ -Probe-Isolating Non-Interference Security.
 

---

**Input** :  $Sh(\mathbf{X})$  – Set of all shares of  $n$  secret variables  $\mathbf{X}$ .  
**Output:**  $\mathbf{Q}$  – Set of  $d + 1$  successful probes.

```

1  $d \leftarrow 1$ 
2 while true do
3   foreach probing set  $\mathbf{Q}$  with  $|\mathbf{Q}| = d$  do
4     simulatable  $\leftarrow$  true
5     foreach  $\mathbf{S} \subseteq Sh(\mathbf{X})^{I_o \cup I_i}$  do
6       if  $p_{\mathbf{Q}, Sh(\mathbf{X})}(\mathbf{1}, \mathbf{1}) \neq p_{\mathbf{Q}, \mathbf{S}}(\mathbf{1}, \mathbf{1}) \cdot p_{\overline{\mathbf{S}}}(\mathbf{1})$  then
7         simulatable  $\leftarrow$  false
8       end
9     end
10    if not simulatable then
11      return  $\mathbf{Q}$ 
12    end
13  end
14   $d \leftarrow d + 1$ 
15 end

```

---

**Algorithmic Verification Approach.** In contrast to NI, for SNI the number of shares per input in each simulation set is bounded by the number of internal probes (instead of the number of all probes). Hence, except for minor difference, the algorithmic verification approach given in Algorithm 3 (notation matching the one given in Definition 6) has the same structure as the approach for NI, but enforcing stronger constraints on the selection of shares (lines 5 and 6) for the simulation set  $\mathbf{S}$ .

#### 4.5 $d$ -Probe-Isolating Non-Interference

For the notion of PINI, the index of any input or output wires correspond to the associated circuit share. In contrast to NI and SNI, the concept of PINI constrains the simulation set not by the number of (internal) probes, but according to the associated circuit shares.

**Verification Approach.** The algorithmic verification approach in order to explore and verify the security notion of PINI for increasing security order  $d$  is given in Algorithm 4. Again, the algorithm is based on the concept of perfect simulatability of every  $\mathbf{Q}$  with a set  $\mathbf{S}$ , in conformity with the notions in Definition 7.

#### 4.6 Uniformity

In order to examine the uniformity of the output sharing of a gadget, we start with the following observation.

**Lemma 3.** *Assume the function  $f$  with single output  $Y \in \mathbb{F}_2$  whose shared version is realized by a gadget with  $d+1$  output shares  $Sh(Y) = (Y^0, \dots, Y^d)$ . The gadget's output sharing is uniform iff any selection of  $d$  output shares make a balanced function.*

*Proof.* We start with  $d = 1$ , i.e., 2 output shares. Let us denote the joint probability of the output shares by  $\rho_{0,0} = Pr[Y^0 = 0, Y^1 = 0]$ ,  $\rho_{0,1} = Pr[Y^0 = 0, Y^1 = 1]$ ,  $\rho_{1,0} = Pr[Y^0 = 1, Y^1 = 0]$ , and  $\rho_{1,1} = Pr[Y^0 = 1, Y^1 = 1]$ , assuming that the gadget's input is uniformly distributed, which is true since the gadget's input sharing should be uniform (essential assumption of Boolean masking, see Section 2.4). Hence, the probability of the output shares can be written as

$$\begin{aligned} Pr[Y^0 = 0] &= \rho_{0,0} + \rho_{0,1}, & Pr[Y^0 = 1] &= \rho_{1,0} + \rho_{1,1}, \\ Pr[Y^1 = 0] &= \rho_{0,0} + \rho_{1,0}, & Pr[Y^1 = 1] &= \rho_{0,1} + \rho_{1,1}. \end{aligned}$$

1) If  $Sh(Y)$  is uniform,  $(Y^0 = 0, Y^1 = 0)$  and  $(Y^0 = 1, Y^1 = 1)$  are equally likely to occur, i.e.,  $\rho_{0,0} = \rho_{1,1}$ . The same holds for  $(Y^0 = 0, Y^1 = 1)$  and  $(Y^0 = 1, Y^1 = 0)$ , i.e.,  $\rho_{0,1} = \rho_{1,0}$ . Therefore,  $Pr[Y^0 = 0] = \rho_{0,0} + \rho_{0,1} = \rho_{1,1} + \rho_{1,0} = Pr[Y^0 = 1]$ , i.e., the gadget's coordinate function  $f^0$  with output  $Y^0$  is balanced. The same trivially holds for  $Y^1$ . Hence, individual balancedness of each output share is essential for uniformity.

2) If the coordinate functions of  $Y^0$  and  $Y^1$  are balanced, we can write

$$\begin{aligned} Pr[Y^0 = 0] = Pr[Y^0 = 1] &\iff \rho_{0,0} + \rho_{0,1} = \rho_{1,0} + \rho_{1,1}, \\ Pr[Y^1 = 0] = Pr[Y^1 = 1] &\iff \rho_{0,0} + \rho_{1,0} = \rho_{0,1} + \rho_{1,1}. \end{aligned}$$

These two equations directly translate into  $\rho_{0,0} = \rho_{1,1}$  and  $\rho_{0,1} = \rho_{1,0}$ . This means that  $(Y^0 = 0, Y^1 = 0)$  and  $(Y^0 = 1, Y^1 = 1)$  are equally likely to occur. The same holds for  $(Y^0 = 0, Y^1 = 1)$  and  $(Y^0 = 1, Y^1 = 0)$ , i.e.,  $Sh(Y)$  is a uniform sharing. Hence, individual balancedness of each output share is also a sufficient condition for uniformity.

For  $d = 2$ , we have  $Sh(Y) = (Y^0, Y^1, Y^2)$ . Assuming a uniform sharing for the gadget's input, similar to the above case for  $d = 1$ , we denote the joint probability of the output shares by  $\rho_{y^0, y^1, y^2} = Pr[Y^0 = y^0, Y^1 = y^1, Y^2 = y^2]$ , e.g.,  $\rho_{1,0,1} = Pr[Y^0 = 1, Y^1 = 0, Y^2 = 1]$ . Exemplary, the joint probability of two output shares  $(Y^0, Y^1)$  can be derived as

$$\begin{aligned} Pr[Y^0 = 0, Y^1 = 0] &= \rho_{0,0,0} + \rho_{0,0,1}, & Pr[Y^0 = 0, Y^1 = 1] &= \rho_{0,1,0} + \rho_{0,1,1}, \\ Pr[Y^0 = 1, Y^1 = 0] &= \rho_{1,0,0} + \rho_{1,0,1}, & Pr[Y^0 = 1, Y^1 = 1] &= \rho_{1,1,0} + \rho_{1,1,1}. \end{aligned} \quad (7)$$

1) In case  $Sh(Y)$  is uniform, we have

$$\rho_{0,0,0} = \rho_{0,1,1} = \rho_{1,0,1} = \rho_{1,1,0}, \quad \rho_{0,0,1} = \rho_{0,1,0} = \rho_{1,0,0} = \rho_{1,1,1}. \quad (8)$$

Hence, it can be trivially seen that

$$Pr[Y^0 = 0, Y^1 = 0] = Pr[Y^0 = 0, Y^1 = 1] = Pr[Y^0 = 1, Y^1 = 0] = Pr[Y^0 = 1, Y^1 = 1],$$

meaning that  $(Y^0, Y^1)$  are jointly balanced. The same holds for other output shares  $(Y^0, Y^2)$  and  $(Y^1, Y^2)$ .

2) If  $(Y^0, Y^1)$  are jointly balanced, all probabilities given in Equation (7) are the same, i.e.,

$$\rho_{0,0,0} + \rho_{0,0,1} = \rho_{0,1,0} + \rho_{0,1,1} = \rho_{1,0,0} + \rho_{1,0,1} = \rho_{1,1,0} + \rho_{1,1,1}.$$

The same can be written for  $(Y^0, Y^2)$  and  $(Y^1, Y^2)$  as

$$\begin{aligned} \rho_{0,0,0} + \rho_{0,1,0} &= \rho_{0,0,1} + \rho_{0,1,1} = \rho_{1,0,0} + \rho_{1,1,0} = \rho_{1,0,1} + \rho_{1,1,1}, \\ \rho_{0,0,0} + \rho_{1,0,0} &= \rho_{0,0,1} + \rho_{1,0,1} = \rho_{0,1,0} + \rho_{1,1,0} = \rho_{0,1,1} + \rho_{1,1,1}. \end{aligned}$$

Combination of these equations leads to the expressions given in Equation (8), indicating the uniformity of  $Sh(Y)$ .

The same procedure can be followed to trivially verify Lemma 3 for  $d > 2$ .

□

**Lemma 4.** *Assume the function  $f$  with  $n$ -bit output  $\mathbf{Y} = (Y_0, \dots, Y_{n-1}) \in \mathbb{F}_2^n$  whose shared version is realized by a gadget with  $d + 1$  output shares  $Sh(\mathbf{Y}) = (\mathbf{Y}^0, \dots, \mathbf{Y}^d)$ . The gadget's output sharing is uniform iff any selection of up to  $n \cdot d$  output shares is balanced excluding the cases where all  $d + 1$  shares of the same output are involved in the selection.*

*Proof.* For  $n = 1$  it is the same as Lemma 3. Hence, we start with  $n = 2$  and minimum number of output shares,  $d + 1 = 2$ . Assuming a uniform sharing for the gadget's input, we denote the joint probability of the output shares by  $\rho_{(y_0^0, y_0^1), (y_1^0, y_1^1)} = Pr[Y_0^0 = y_0^0, Y_0^1 = y_0^1, Y_1^0 = y_1^0, Y_1^1 = y_1^1]$ , e.g.,  $\rho_{(1,0), (1,1)} = Pr[Y_0^0 = 1, Y_0^1 = 0, Y_1^0 = 1, Y_1^1 = 1]$ .

Exemplary, the joint probability of two output shares  $(Y_0^0, Y_1^1)$  is written as

$$\begin{aligned} Pr[Y_0^0 = 0, Y_1^1 = 0] &= \rho_{(0,0), (0,0)} + \rho_{(0,0), (1,0)} + \rho_{(0,1), (0,0)} + \rho_{(0,1), (1,0)}, \\ Pr[Y_0^0 = 0, Y_1^1 = 1] &= \rho_{(0,0), (0,1)} + \rho_{(0,0), (1,1)} + \rho_{(0,1), (0,1)} + \rho_{(0,1), (1,1)}, \\ Pr[Y_0^0 = 1, Y_1^1 = 0] &= \rho_{(1,0), (0,0)} + \rho_{(1,0), (1,0)} + \rho_{(1,1), (0,0)} + \rho_{(1,1), (1,0)}, \\ Pr[Y_0^0 = 1, Y_1^1 = 1] &= \rho_{(1,0), (0,1)} + \rho_{(1,0), (1,1)} + \rho_{(1,1), (0,1)} + \rho_{(1,1), (1,1)}. \end{aligned} \tag{9}$$

1) If  $Sh(\mathbf{Y})$  is uniform, we have

$$\begin{aligned} \rho_{(0,0), (0,0)} &= \rho_{(0,0), (1,1)} = \rho_{(1,1), (0,0)} = \rho_{(1,1), (1,1)}, \\ \rho_{(0,0), (0,1)} &= \rho_{(0,0), (1,0)} = \rho_{(1,1), (0,1)} = \rho_{(1,1), (1,0)}, \\ \rho_{(0,1), (0,0)} &= \rho_{(0,1), (1,1)} = \rho_{(1,0), (0,0)} = \rho_{(1,0), (1,1)}, \\ \rho_{(0,1), (0,1)} &= \rho_{(0,1), (1,0)} = \rho_{(1,0), (0,1)} = \rho_{(1,0), (1,0)}. \end{aligned}$$

This results in equal probabilities for all probabilities given in Equation (9), such that

$$Pr[Y_0^0 = 0, Y_1^1 = 0] = Pr[Y_0^0 = 0, Y_1^1 = 1] = Pr[Y_0^0 = 1, Y_1^1 = 0] = Pr[Y_0^0 = 1, Y_1^1 = 1],$$

i.e., the two output shares  $(Y_0^0, Y_1^1)$  are jointly balanced. The same can be similarly verified all other combinations  $(Y_0^0, Y_1^0)$ ,  $(Y_0^1, Y_1^0)$ , and  $(Y_0^1, Y_1^1)$ .

2) If  $(Y_0^0, Y_1^1)$  are jointly balanced, based on Equation (9) we have

$$\begin{aligned}
&\rho_{(0,0),(0,0)} + \rho_{(0,0),(1,0)} + \rho_{(0,1),(0,0)} + \rho_{(0,1),(1,0)} = \\
&\rho_{(0,0),(0,1)} + \rho_{(0,0),(1,1)} + \rho_{(0,1),(0,1)} + \rho_{(0,1),(1,1)} = \\
&\rho_{(1,0),(0,0)} + \rho_{(1,0),(1,0)} + \rho_{(1,1),(0,0)} + \rho_{(1,1),(1,0)} = \\
&\rho_{(1,0),(0,1)} + \rho_{(1,0),(1,1)} + \rho_{(1,1),(0,1)} + \rho_{(1,1),(1,1)} = 1/4.
\end{aligned} \tag{10}$$

This leads to

$$\begin{aligned}
&\rho_{(0,0),(0,0)} + \rho_{(0,0),(1,0)} + \rho_{(0,1),(0,0)} + \rho_{(0,1),(1,0)} \\
&\quad + \rho_{(0,0),(0,1)} + \rho_{(0,0),(1,1)} + \rho_{(0,1),(0,1)} + \rho_{(0,1),(1,1)} = \\
&\rho_{(1,0),(0,0)} + \rho_{(1,0),(1,0)} + \rho_{(1,1),(0,0)} + \rho_{(1,1),(1,0)} \\
&\quad + \rho_{(1,0),(0,1)} + \rho_{(1,0),(1,1)} + \rho_{(1,1),(0,1)} + \rho_{(1,1),(1,1)} = 1/2,
\end{aligned} \tag{11}$$

meaning that  $Pr[Y_0^0 = 0] = Pr[Y_0^0 = 1]$ , i.e., it is balanced. The same can be written for  $Pr[Y_1^1 = 0] = Pr[Y_1^1 = 1]$ , and similarly for  $(Y_0^0, Y_1^0)$ ,  $(Y_0^1, Y_1^0)$ , and  $(Y_0^1, Y_1^1)$ . In short, every single output bit is balanced, hence, according to Lemma 3, the sharing of every output is individually uniform. Note that, in general when a function with  $d$  output bits is balanced, any combination of  $d' < d$  output bits also makes a balanced function [34, §12.1.2].

According to Equation (9) and Equation (10), we exemplarily write

$$Pr[Y_0^0 = 0, Y_1^1 = 0] = 1/4.$$

On the other hand, according to Equation (11) we have

$$Pr[Y_0^0 = 0] = 1/2, \quad Pr[Y_1^1 = 0] = 1/2,$$

which implies  $Pr[Y_0^0 = 0, Y_1^1 = 0] = Pr[Y_0^0 = 0] \cdot Pr[Y_1^1 = 0]$ . The same can similarly be seen for  $(Y_0^0, Y_1^1) = (0, 1)$ ,  $(1, 0)$  and  $(1, 1)$ , meaning that

$$Pr[Y_0^0, Y_1^1] = Pr[Y_0^0] \cdot Pr[Y_1^1]. \tag{12}$$

In other words,  $Y_0^0$  and  $Y_1^1$  are statistically independent. In a similar way, statistical independence of  $(Y_0^0, Y_1^0)$ ,  $(Y_0^1, Y_1^0)$ , and  $(Y_0^1, Y_1^1)$  can be shown.

Now, let us denote conditional probability  $Pr[Y_0^0 = y_0^0, Y_0^1 = y_0^1, Y_1^0 = y_1^0, Y_1^1 = y_1^1 | Y_0 = y_0, Y_1 = y_1]$  by  $\rho_{(y_0^0, y_0^1), (y_1^0, y_1^1) | (y_0, y_1)}$ . For the sharing  $Sh(\mathbf{Y})$  to be uniform, according to Equation (4) and exemplarily for  $\mathbf{Y} = (0, 0)$  we should have

$$\rho_{(0,0),(0,0)|(0,0)} = \rho_{(0,0),(1,1)|(0,0)} = \rho_{(1,1),(0,0)|(0,0)} = \rho_{(1,1),(1,1)|(0,0)} = 1/4. \tag{13}$$

The same should hold for other values of  $\mathbf{Y} = (0, 1)$ ,  $(1, 0)$ , and  $(1, 1)$ . Considering the statistical independence of  $(Y_0^1, Y_1^0)$  explained above, We can write

$$\begin{aligned}
\rho_{(0,0),(0,0)|(0,0)} &= Pr[Y_0^0=0, Y_0^1=0, Y_1^0=0, Y_1^1=0 | Y_0=0, Y_1=0] \\
&= Pr[Y_0^0=0, Y_1^1=0 | Y_0^1=0, Y_1^0=0, Y_0=0, Y_1=0] \cdot Pr[Y_0^1=0, Y_1^0=0 | Y_0=0, Y_1=0] \\
&= 1 \cdot Pr[Y_0^1=0 | Y_0=0, Y_1=0] \cdot Pr[Y_1^0=0 | Y_0=0, Y_1=0]
\end{aligned}$$



Due to the balancedness of every individual output, we have

$$\Pr[Y_0^1 = 0 | Y_0 = 0, Y_1 = 0] = \Pr[Y_0^1 = 0 | Y_0 = 0, Y_1 = 0] = 1/2.$$

This leads to  $\rho_{(0,0),(0,0)|(0,0)} = 1/4$ . The same can be shown for  $\rho_{(0,0),(1,1)|(0,0)}$ ,  $\rho_{(1,1),(0,0)|(0,0)}$ , and  $\rho_{(1,1),(1,1)|(0,0)}$ , satisfying Equation (13). The same can be similarly verified for  $\mathbf{Y} = (0, 1)$ ,  $(1, 0)$ , and  $(1, 1)$ , hence the uniformity of  $Sh(\mathbf{Y})$ .

The same procedure can be followed to verify Lemma 4 for  $n > 2$  and  $d > 2$ .  $\square$

Indeed, for a given circuit netlist we efficiently perform balancedness checks directly based on the ROBDDs of the circuit.

## 5 Related Work

For formal verification of masked implementations, both in software and hardware, several tools and frameworks have been proposed, each following a different methodology and verification approach.

**Formal Verification of Software Implementations.** For automated masking of software implementations, the work of Moss *et al.* [38] was first to consider a type-based methodology for security annotation while dynamically repairing the masked implementation based on heuristics if leakage was detected at some point in the program flow. As any type-based approach inevitably results in an overly conservative verification, logic-based methods have been proposed as an alternative approach. Here, the work by Byrak *et al.* Byrak [6] translates verification to a set of Boolean satisfiability problems which can then be solved by a SAT solver. Nonetheless, both approaches only consider verification of masking against first-order attacks.

Later, an SMT-solver-based method for formally verifying even higher-order security has been introduced in [24]. As for [6], this verification method is also based on the notion of *perfect masking* as presented in [13]. Similarly, in [44] another method for verifying perfect masking was introduced, this time aiming to optimize the trade-off between accuracy (as offered by logic-based approaches) and efficiency (as given in type-based verification). Eventually, a composition-based verification approach in direct conformity with  $d$ -probing security (i.e., without any false negatives) is given by Belaïd *et al.* in [9].

**Formal Verification of Hardware Implementations.** Considering hardware designs, the work of Bloem *et al.* [11,12] resulted in a seminal tool enabling formal verification even in the presence of glitches, but with restriction to verification of probing security only. Most recently, the work of Cassiers *et al.* [16] proposes a composition-based approach of verifying probing security of a concrete implementation composed of so-called *Hardware Private Circuits*.

Besides, the latest version of `maskVerif` – as presented in [2] – supports efficient verification of  $d$ -probing security,  $d$ -NI, and  $d$ -SNI for arbitrary orders for both software and hardware designs, even in the presence of glitches. Currently, `maskVerif` is the state-of-the-art tool offering the widest-ranging verification features which is not composition-based, hence, in the following we provide a more detailed discussion and comparison to our developed tool.

### 5.1 Comparison to `maskVerif`

In general, `maskVerif` offers an efficient approach to verify security of masked software and hardware implementations. In contrast to our approach, `maskVerif` utilizes a symbolic representation of leakage defined by a given syntax and semantic. For verifying security, the tool first assigns a symbolic leakage set to every instruction. Depending on the security order, each combination of symbolic leakage sets, i.e., each possible observation, is exploited afterwards and tested for the absence of secret dependency through performing a syntactical check and applying semantic-preserving transformation on the sets.

Due to its language-based verification approach, security checks in `maskVerif` follow a very conservative approach for particular designs. More precisely, it may falsely reject some secure designs because the checks are not based on explicit statistical properties in conformity with the actual definition of the security notions. Due to these limitations of a purely syntactical verification, it more likely fails to provide correct verification of probing security if an output of a masked circuit is not non-complete (as used for TIs) but also does not rely on fresh randomness. In other words, its computation is a result of all input shares of at least one input without using any fresh randomness for blinding purposes. In particular, a computation using all input shares not necessarily implies statistical dependency on the corresponding input (e.g., due to blinding with shares of different inputs). Nonetheless, since the verification approach of `maskVerif` is mainly based on syntactical checks, it may falsely categorize the design as not being probing secure although it is (i.e., resulting in false negative).

**Examples for False Negatives in `maskVerif`.** One small example is a shared version of the 4-bit bijection quadratic class  $\mathcal{Q}_{12}^4$  (based on the classification given in [10]), utilizing two shares per input, as presented in Appendix of [42]. Using `maskVerif`, this design is falsely categorized as not being first-order probing secure although all possible probes are statistically independent of the secrets. Hence, according to `maskVerif`, in order to gain successful verification, one possible solution would be to introduce additional randomness  $r \in \mathbb{F}_2$  into the design, such that:

$$\begin{aligned} x_1 &= F_1(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = a_1 \\ x_2 &= F_2(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = a_2 \\ y_1 &= G_1(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = a_1 c_1 \oplus b_1 \oplus r & \bar{x}_1 &= x_1 \end{aligned}$$

$$\begin{aligned}
y_2 &= G_2(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = a_1 c_2 & \bar{x}_2 &= x_2 \\
y_3 &= G_3(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = a_2 c_1 \oplus b_2 \oplus r & \bar{y}_1 &= y_1 \oplus y_2 \\
y_4 &= G_4(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = a_2 c_2 & \bar{y}_2 &= y_3 \oplus y_4 \\
z_1 &= H_1(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = a_1 b_1 \oplus a_1 c_1 \oplus c_1 & \bar{z}_1 &= z_1 \oplus z_2 \\
z_2 &= H_2(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = a_1 b_2 \oplus a_1 c_2 \oplus r & \bar{z}_2 &= z_3 \oplus z_4 \\
z_3 &= H_3(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = a_2 b_1 \oplus a_2 c_1 \oplus r & \bar{t}_1 &= t_1 \\
z_4 &= H_4(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = a_2 b_2 \oplus a_2 c_2 \oplus c_2 & \bar{t}_2 &= t_2 \\
t_1 &= K_1(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = d_1 \\
t_2 &= K_2(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = d_2
\end{aligned}$$

This new realization of  $\mathcal{Q}_{12}^4$  is now correctly verified by `maskVerif` as being first-order probing secure. However, introducing randomness is costly and not necessary to gain independence of the secret input, i.e., fulfilling first-order probing security.

However, this given example based on  $\mathcal{Q}_{12}^4$  is only a small design. For larger and more complex circuits, this inaccurate determination of the security level will lead to significantly more overhead being introduced during the design process. An example for a more complex design, which is falsely classified as not being first-order probing secure, is the PRESENT S-box realized as a TI utilizing three shares for every output and input bit as presented in [40].

In fact, in order to achieve a sufficient security level while only introducing marginal overhead into the design, it is thus necessary to be in conformity with the security notions. As our verification is based on actual statistical properties between probes and inputs, i.e., in accordance with the formal definitions of the security notions, we actually meet this need and completely avoid false negatives. This eventually is expected to result in less overhead in terms of area and randomness when designing and implementing masked implementations. Moreover, and in addition to features in `maskVerif`, our tool is extended to verify  $d$ th-order PINI and the output uniformity of a given design while also returning the first probe combination found which is not in conformity with the respective security notion.

Hence, despite being slower and slightly less efficient for larger design compared to a type-based approach, as used for instance in `maskVerif`, our tool is assumed to close the gap between accuracy and efficiency by providing a complete and sound verification framework for the security and composability of both software and hardware designs.

## 6 Experiments and Evaluations

This section presents implementation, evaluation, and performance results of our proposed tool for formal verification of masked circuits.

---

<https://github.com/chair-for-security-engineering/silver>

**Implementation.** For a practical evaluation of our proposed concepts and methodologies, we opted to implement a formal verification tool using `Sylvan` [22], a state-of-the-art BDD high-performance, multi-core decision diagram package implemented in C/C++. Further, we also customized and extended the native instructions of `Sylvan` in order to provide and support dedicated operations computing  $p_{\mathbf{X}}(\mathbf{1})$  and  $p_{\mathbf{X},\mathbf{Y}}(\mathbf{1}, \mathbf{1})$  based on [35], i.e., without formal construction of new BDDs each time these operations are executed. Eventually, our framework implements all verification algorithms presented in Section 4 for both, standard and robust probing model, and is running in a 64-bit Linux Operating System (OS) environment on an Intel Xeon E5-1660v4 CPU with a clock frequency of 3.20 GHz and 128 GB of Random-Access Memory (RAM).

Our tool process a netlist file as the specification of the CUT. The user can either make such a netlist manually, e.g., for software applications or a sequence of operations, or can provide a verilog file as the result of a hardware synthesis, e.g., Design Compiler or Yosys, using a restricted library (defined in Section 2.3). It is beneficial to directly evaluate the circuit’s netlist as any user-originated mistakes or flaws (e.g., not keeping design hierarchy, hence violating non-completeness [39]) can be detected.

**Experiments and Benchmarks.** In Table 1, we summarize verification and performance results for our tool using various different examples as a benchmark. For this, the number  $d$  indicates the masking order of the circuit design (i.e., the number of input shares given as  $d + 1$ ), while the number next to the tick indicates the maximum security order found by our tool during security check and verification (i.e., the number of probes that did not lead to a failing check). For all designs, we provide analysis results for the security notions of  $d$ -probing, NI, SNI, PINI, and uniformity of the output sharing. Except for uniformity, all security checks are performed for the standard (i.e., without physical defaults in terms of *glitches*) and robust (i.e., with *glitches*) leakage models as presented in Section 3. Eventually, along with the number of potential probe positions, i.e., the number of distinct wires determined by the number of gates in the circuit, the security parameter  $d$  yields the verification complexity in terms of possible observations  $\mathcal{O} = \sum_{i=1}^d \binom{pos}{i}$ .

**Examples.** In Table 1, we list verification results for three different categories of masked circuits. In the first category, denoted as **Gadgets**, we analyze different variants to implement a masked field multiplication for  $\mathbb{F}_2$ . Note, that for the SNI variant of Domain-Oriented Masking (DOM) multiplier [29], we simply added additional registers at the output to achieve an SNI-secure circuit. Interestingly, PARA1 [5] and PARA2 gadgets are up to  $d$ -SNI secure in both models, but higher-order variants cannot achieve full security, and need design modifications instead (although still SNI for smaller  $d$ ). We should stress that `maskVerif` reports PARA3 to be not SNI, while it is up to 2-SNI, which is correctly reported

**Table 1.** Verification of Various Masked Circuits and Security Notions.

Scheme	Pos. <sup>†</sup>	$d$	Probing		NI		SNI		PINI		Unif.
			std.	rob.	std.	rob.	std.	rob.	std.	rob.	
<b>Gadgets</b>											
DOM [29]	19	1	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✗[0 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]
DOM [29]	42	2	✓[3 ms]	✓[4 ms]	✓[6 ms]	✓[19 ms]	✓[8 ms]	✗[0 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]
DOM [29]	74	3	✓[98 ms]	✓[1.2 s]	✓[2.2 s]	✓[23.7 s]	✓[3.2 s]	✗[0 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]
DOM SNI [26]	21	1	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]
DOM SNI [26]	45	2	✓[3 ms]	✓[5 ms]	✓[6 ms]	✓[30 ms]	✓[7 ms]	✓[29 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]
DOM SNI [26]	78	3	✓[0.1 s]	✓[1.5 s]	✓[2.4 s]	✓[39.4 s]	✓[3.7 s]	✓[39.4 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
PARA1 [5]	22	1	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]
PARA2 [5]	45	2	✓[3 ms]	✓[6 ms]	✓[5 ms]	✓[32 ms]	✓[8 ms]	✓[37 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]
PARA3 [5]	68	3	✓[61 ms]	✓[0.5 s]	✓[1.2 s]	✓[12.1 s]	✗[0.6 s]	✗[0 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]
PARA3 SNI [5]	82	3	✓[0.2 s]	✓[1.4 s]	✓[2.8 s]	✓[35.5 s]	✓[4.1 s]	✓[40.4 s]	✗[0 ms]	✗[0 ms]	✓[0 ms]
PINI1 [17]	21	1	✓[0 ms]	✗[0 ms]	✓[0 ms]	✗[0 ms]	✓[0 ms]	✗[0 ms]	✓[0 ms]	✗[0 ms]	✓[0 ms]
PINI2 [17]	51	2	✓[7 ms]	✗[0 ms]	✓[10 ms]	✗[0 ms]	✓[12 ms]	✗[0 ms]	✓[22 ms]	✗[0 ms]	✓[0 ms]
HPC1 [16]	22	1	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✗[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]
HPC1 [16]	52	2	✓[5 ms]	✓[7 ms]	✓[7 ms]	✓[23 ms]	✓[9 ms]	✗[0 ms]	✓[16 ms]	✓[46 ms]	✓[0 ms]
HPC2 [16]	32	1	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✗[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]
HPC2 [16]	75	2	✓[6 ms]	✓[12 ms]	✓[1.1 ms]	✓[37 ms]	✓[1.3 ms]	✗[0 ms]	✓[19 ms]	✓[61 ms]	✓[0 ms]
ISW SNI REF [26]	26	1	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]
ISW SNI REF [26]	65	2	✓[5 ms]	✓[8 ms]	✓[7 ms]	✓[36 ms]	✓[9 ms]	✓[34 ms]	✓[16 ms]	✓[0 ms]	✓[0 ms]
CMS3 [36]	104	3	✗[0.1 s]	✗[0.3 s]	✗[0.8 s]	✗[2.6 s]	✗[1.3 s]	✗[4.4 s]	✗[0 ms]	✗[0 ms]	✓[0 ms]
UMA2 [36]	81	2	✗[2 ms]	✗[0 ms]	✗[7 ms]	✗[4 ms]	✗[6 ms]	✗[3 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]
DOM2 DEP <sup>‡</sup> [36]	56	2	✓[4 ms]	✗[8 ms]	✓[3 ms]	✗[20 ms]	✓[4 ms]	✗[0 ms]	✓[4 ms]	✗[21 ms]	✓[0 ms]
<b>S-boxes</b>											
PRESENT <sub>TI</sub> [40]	177	2	✓[4 ms]	✓[8 ms]	✗[4 ms]	✗[0 ms]	✗[3 ms]	✗[0 ms]	✗[0 ms]	✗[0 ms]	✓[2 ms]
PRESENT <sub>TI</sub> [25]	377	2	✓[15 ms]	✓[6 ms]	✗[2 ms]	✗[0 ms]	✗[2 ms]	✗[0 ms]	✗[1 ms]	✗[0 ms]	✓[0 ms]
PRESENT <sub>TI</sub> [25]	161	2	✗[3 ms]	✗[4 ms]	✗[32 ms]	✗[0 ms]	✗[26 ms]	✗[0 ms]	✗[2 ms]	✗[0 ms]	✗[0 ms]
PRINCE <sub>TI</sub> [37]	150	2	✓[2 ms]	✓[10 ms]	✗[2 ms]	✗[0 ms]	✗[2 ms]	✗[0 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]
PRINCE <sub>CMS</sub> [14]	261	1	✓[3 ms]	✓[97 ms]	✓[7 ms]	✓[2.8 s]	✓[9 ms]	✓[1 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]
SKINNY8 <sub>TI</sub> [7]	240	2	✓[51.2 s]	✓[2 min]	✗[2 min]	✗[2.0 s]	✗[2 min]	✗[2.0 s]	✗[77 ms]	✗[1.3 s]	✓[29.6 s]
SKINNY8 <sub>CMS</sub> [8]	192	1	✓[20 ms]	✓[0.3 s]	✗[0.3 s]	✗[17 ms]	✗[0.3 s]	✗[15 ms]	✗[1 ms]	✗[1 ms]	✓[1 ms]
AES <sub>DOM</sub> [29]	884	1	✓[3.3 s]	✓[21 min]	✗[0.8 s]	✗[0.4 s]	✗[0.8 s]	✗[0.4 s]	✗[0.2 s]	✗[40 ms]	✓[0.1 s]
AES <sub>CMS</sub> [19]	938	1	✓[9.4 s]	✓[2.9 h]	✗[0.9 s]	✗[0.5 s]	✗[0.9 s]	✗[0.5 s]	✗[0.2 s]	✗[42 ms]	✓[1.8 s]
<b>Functions</b>											
A <sub>in</sub> [37]	18	1	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]
A <sub>m</sub> [37]	20	1	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]
A <sub>out</sub> [37]	20	1	✓[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]	✓[0 ms]	✓[0 ms]
Q <sub>12</sub> <sup>‡</sup> [42]	48	1	✓[0 ms]	✓[0 ms]	✗[0 ms]	✗[0 ms]	✗[0 ms]	✗[0 ms]	✗[0 ms]	✗[0 ms]	✓[0 ms]

<sup>†</sup> Number of possible probe positions, i.e., output wires of gates. <sup>‡</sup> Assuming identical inputs, i.e.,  $a = b$ .

by our tool. Also, our tool could identify and report all flaws described in [36] including the probes as identified by the authors. Our second category lists different masked **S-boxes** of lightweight and standard block ciphers implemented following the concepts of Consolidating Masking Schemes (CMS) [42], TI [39], or DOM [29]. Eventually, our last category **Functions** lists arbitrary masked functions with linear or quadratic algebraic complexity.

Interestingly, besides the linear functions, only the Hardware Private Circuit (HPC) gadgets [16] and the ISW-SNI gadget [26] extended by an additional refresh of one input are secure in the robust, glitch-extended probing model

under the notion of PINI. Since PINI gadgets [17] are not robust probing secure, they are mainly useful in software applications (i.e., standard probing model). Indeed, since all HPC gadgets are secure under the PINI notion (for both probing models) and can be composed trivially, security under the SNI notion is no longer compulsory (as confirmed by our evaluation results for the robust, glitch-extended probing model). Also, besides  $\mathcal{Q}_{12}^4$  we also analyzed other quadratic functions provided in [42] and our tool revealed that the implementation of  $\mathcal{Q}_{300}^4$  as given by the authors is not uniform.

**Verification Complexity.** In contrast to the language-based verification approach of `maskVerif`, our framework heavily relies on statistical independence verification of probability distributions in order to avoid false negatives. Therefore, the overall run time of our verification approach is mainly governed by construction of intermediate ROBDDs representing the logical conjunctions as part of the statistical independence checks for the security notions. As already shown in [41], the complexity of constructing ROBDDs increases mainly by the number of product terms occurring in the minimal Disjunctive Normal Form (DNF) of the represented Boolean function.

Generally speaking, when considering higher-order security verification, we have to test for statistical independence of larger sets of random variables with possible non-linear dependence on many of the inputs. As our test of statistical independence is based on logical conjunctions of sets of random variables (and every possible subset), this leads to a high number of product terms occurring in the resulting DNF, and hence to an increased complexity of the constructed ROBDDs. As a result, verification speed of our framework is mainly influenced by the complexity, i.e., input dependencies of wires, and the maximum security order of the CUT.

Further, with increasing security order, the combinatorial complexity  $\mathcal{O}$  of constructing all possible observations grows exponentially. However, as we opted for accurate security verification without relying on heuristics, reducing the number of probe combinations is not trivial, but instead we have to check and verify all of them. Although some joint distributions might be similar for different probe combinations, we still have to analyze most combinations which is rather time consuming for higher security orders and larger circuits. It is worth to mention that if any of the combinations leads to a negative statistical independence, the tool stops and reports the found leaking probes. Hence, the maximum run time is taken only if the CUT passes all desired security checks.

## 7 Conclusion

In this work, we developed and presented a sound and accurate framework to verify the security and composability of masked gate-level netlists and circuits

---

This case is caught by the internal caching scheme of the `Sylvan` BDD package which first checks if the current operation has been performed and cached recently before executing the actual operation in case no cache entry was found.

directly resulting from hardware logic synthesis processes. In particular, our approach enables formal verification of all pertinent security notions in the domain of physical security and is applicable to both, software and hardware designs, even considering physical defaults in terms of *glitches*. More concretely, it supports sound, accurate, and immediate verification whether a masked implementation provides probing security, Non-Interference, Strong Non-Interference, and Probe-Isolating Non-Interference – even for higher security orders. In addition, we proposed and integrated a novel methodology of verifying uniformity of the output sharing of a masked gadget. Eventually, if verification fails, it reports the failing set of probes being in non-conformity with the corresponding security notion.

In contrast to common type-based methods, our approach is based on formal verification of statistical properties in direct conformity with the fundamental definitions of the security notions. As a result, our approach completely avoids overly conservative decisions when falsely declaring designs as not being secure (false negatives), ultimately leading to a reduction in design overhead as otherwise introduced by additional (and expensive) fresh randomness. For this, all verification checks of statistical properties are executed efficiently by reducing statistical independence checks on joint distributions over multiple binary random variables to checks of distributions over single binary random variables, which can be efficiently done utilizing the concepts of ROBDDs. Eventually, this results in a framework exceeding comparable tools in accuracy and functionality while still being reasonable efficient for most applications and common use cases.

The current version of our tool is mainly beneficial to evaluate gadgets, particularly at higher orders, although we have given its capability to examine the entire S-boxes (see Table 1). For future work, we will focus on extending capabilities and improving efficiency of our tool, mainly with respect to larger and more complex circuits and implementations and higher security orders. For this, distinguishing univariate and multivariate leakages would be interesting, as it would allow *divide-and-conquer* approaches based on partitioning complex circuits along register stages while security analysis then would be performed on smaller circuits automatically. Certainly, verification then can be performed more efficiently, even for large and complex designs and higher-orders as long as the design is not entirely combinational but contains register stages. The future version of our tool should receive the netlist of a complete cipher implementation, unroll the loops, divide it into separate gadgets, and conduct security evaluation respectively.

## Acknowledgments

The work described in this paper has been supported in part by the German Research Foundation (DFG) under Germany’s Excellence Strategy - EXC 2092 CASA - 390781972, and through the project 393207943 “Security for Internet of Things with Low Energy and Low Power Consumption (GreenSec).

## References

1. Akers, S.B.: Binary Decision Diagrams. *IEEE Trans. Computers* (1978)
2. Barthe, G., Belaïd, S., Cassiers, G., Fouque, P., Grégoire, B., Standaert, F.: maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In: *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Proceedings, Part I*. LNCS, vol. 11735, pp. 300–318. Springer (2019)
3. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P.: Verified Proofs of Higher-Order Masking. In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part I*. LNCS, vol. 9056, pp. 457–485. Springer (2015)
4. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. pp. 116–129. ACM (2016)
5. Barthe, G., Dupressoir, F., Faust, S., Grégoire, B., Standaert, F., Strub, P.: Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part I*. LNCS, vol. 10210, pp. 535–566 (2017)
6. Bayrak, A.G., Regazzoni, F., Novo, D., Ienne, P.: Sleuth: Automated Verification of Software Power Analysis Countermeasures. In: *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Proceedings*. LNCS, vol. 8086, pp. 293–310. Springer (2013)
7. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Proceedings, Part II*. LNCS, vol. 9815, pp. 123–153. Springer (2016)
8. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: SKINNY-AEAD and SKINNY-Hash. *IACR Trans. Symmetric Cryptol.* (2020)
9. Belaïd, S., Goudarzi, D., Rivain, M.: Tight Private Circuits: Achieving Probing Security with the Least Refreshing. In: *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Part II*. LNCS, vol. 11273, pp. 343–372. Springer (2018)
10. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Tokareva, N.N., Vitkup, V.: Threshold implementations of small s-boxes. *Cryptogr. Commun.* (2015)
11. Bloem, R., Groß, H., Iusupov, R., Könighofer, B., Mangard, S., Winter, J.: Formal Verification of Masked Hardware Implementations in the Presence of Glitches. In: *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II*. LNCS, vol. 10821, pp. 321–353. Springer (2018)
12. Bloem, R., Groß, H., Iusupov, R., Krenn, M., Mangard, S.: Sharing independence & relabeling: Efficient formal verification of higher-order masking. *IACR Cryptol. ePrint Arch.* (2018)



13. Blömer, J., Guajardo, J., Krummel, V.: Provably secure masking of AES. *IACR Cryptol. ePrint Arch.* (2004)
14. Bozilov, D., Knezevic, M., Nikov, V.: Optimized threshold implementations: Minimizing the latency of secure cryptographic accelerators. In: *Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Revised Selected Papers*. LNCS, vol. 11833, pp. 20–39. Springer (2019)
15. Bryant, R.E.: *Graph-Based Algorithms for Boolean Function Manipulation*. IEEE Trans. Computers (1986)
16. Cassiers, G., Grégoire, B., Levi, I., Standaert, F.: Hardware private circuits: From trivial composition to full verification. *IACR Cryptol. ePrint Arch.* (2020)
17. Cassiers, G., Standaert, F.: Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Information Forensics and Security* (2020)
18. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Proceedings*. LNCS, vol. 1666, pp. 398–412. Springer (1999)
19. Cnudde, T.D., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., Rijmen, V.: Masking AES with  $d+1$  shares in hardware. In: *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Proceedings*. LNCS, vol. 9813, pp. 194–212. Springer (2016)
20. Coron, J.: Formal Verification of Side-Channel Countermeasures via Elementary Circuit Transformations. In: *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Proceedings*. LNCS, vol. 10892, pp. 65–82. Springer (2018)
21. De Meyer, L., Bilgin, B., Reparaz, O.: Consolidating Security Notions in Hardware Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2019)
22. van Dijk, T.: *Sylvan: multi-core decision diagrams*. Ph.D. thesis, University of Twente, Enschede, Netherlands (2016)
23. Duc, A., Dziembowski, S., Faust, S.: Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In: *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*. LNCS, vol. 8441, pp. 423–440. Springer (2014)
24. Eldib, H., Wang, C., Schaumont, P.: Formal Verification of Software Countermeasures against Side-Channel Attacks. *ACM Trans. Softw. Eng. Methodol.* (2014)
25. Ender, M., Ghandali, S., Moradi, A., Paar, C.: The first thorough side-channel hardware trojan. In: *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Proceedings, Part I*. LNCS, vol. 10624, pp. 755–780. Springer (2017)
26. Faust, S., Grosso, V., Pozo, S.M.D., Paglialonga, C., Standaert, F.: Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2018)
27. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Proceedings*. LNCS, vol. 2162, pp. 251–261. Springer (2001)
28. Groß, H., Mangard, S.: A Unified Masking Approach. *J. Cryptographic Engineering* (2018)
29. Groß, H., Mangard, S., Korak, T.: An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In: *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA,*

- USA, February 14-17, 2017, Proceedings. LNCS, vol. 10159, pp. 95–112. Springer (2017)
30. Hutter, M., Schmidt, J.: The Temperature Side Channel and Heating Fault Attacks. In: Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Revised Selected Papers. LNCS, vol. 8419, pp. 219–235. Springer (2013)
  31. Ishai, Y., Sahai, A., Wagner, D.A.: Private Circuits: Securing Hardware against Probing Attacks. In: Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Proceedings. LNCS, vol. 2729, pp. 463–481. Springer (2003)
  32. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Proceedings. LNCS, vol. 1109, pp. 104–113. Springer (1996)
  33. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Proceedings. LNCS, vol. 1666, pp. 388–397. Springer (1999)
  34. Mesnager, S.: Bent Functions - Fundamentals and Results. Springer (2016)
  35. Miller, D.M.: An improved method for computing a generalized spectral coefficient. IEEE Trans. on CAD of Integrated Circuits and Systems (1998)
  36. Moos, T., Moradi, A., Schneider, T., Standaert, F.: Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed. IACR Trans. Cryptogr. Hardw. Embed. Syst. (2019)
  37. Moradi, A., Schneider, T.: Side-Channel Analysis Protection and Low-Latency in Action - - Case Study of PRINCE and Midori -. In: Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Part I. LNCS, vol. 10031, pp. 517–547 (2016)
  38. Moss, A., Oswald, E., Page, D., Tunstall, M.: Compiler Assisted Masking. In: Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Proceedings. LNCS, vol. 7428, pp. 58–75. Springer (2012)
  39. Nikova, S., Rijmen, V., Schl affer, M.: Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. J. Cryptology (2011)
  40. Poschmann, A., Moradi, A., Khoo, K., Lim, C., Wang, H., Ling, S.: Side-channel resistant crypto for less than 2, 300 GE. J. Cryptology (2011)
  41. Raseen, M., Prasad, P.W.C., Assi, A.: An efficient estimation of the ROBDD's complexity. Integr. (2006)
  42. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating Masking Schemes. In: Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Proceedings, Part I. LNCS, vol. 9215, pp. 764–783. Springer (2015)
  43. Trichina, E.: Combinational Logic Design for AES SubByte Transformation on Masked Data. IACR Cryptol. ePrint Arch. (2003)
  44. Zhang, J., Gao, P., Song, F., Wang, C.: SCInfer: Refinement-Based Verification of Software Countermeasures Against Side-Channel Attacks. In: Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Proceedings, Part II. LNCS, vol. 10982, pp. 157–177. Springer (2018)

## 4.2 Transitional Leakage in Theory and Practice – Unveiling Security Flaws in Masked Circuits

### Publication Data

Nicolai Müller, David Knichel, Pascal Sasdrich, and Amir Moradi. Transitional Leakage in Theory and Practice – Unveiling Security Flaws in Masked Circuits. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):266–288, 2022

*This work is reproduced here with permission. This work is licensed under a Creative Commons Attribution 4.0 International License. Copyright is held by the authors. The author of this thesis is also an author of this research paper.*

**Content.** In this work, we extend the original version of SILVER [KSM20] to be able to cope with transitions in iterative circuits. For this, we derive an appropriate methodology to cover transitions in a scenario where one and the same physical module is executed several times in a loop and it is possible to give a sequence of different inputs to the circuit. As it is the most relevant case for hardware, we focus on the simultaneous occurrence of glitches *and* transitions within the circuit. By performing extensive case studies, we show that, while considering transitions is crucial for guaranteeing practical security, adding extensive overhead into the design to counteract leakage originating from transitions is not necessary in most of the cases.

**Contribution.** The author was involved in developing the methodology and delivering case studies where transitions can be delicate to handle. The author was further involved in providing hardware modules and discuss the relevance of transitions in the context of composable gadgets. Last but not least, the author significantly contributed to the writing of the publication. The author likes to thank all co-authors for their substantial contribution.



# Transitional Leakage in Theory and Practice

## Unveiling Security Flaws in Masked Circuits

Nicolai Müller<sup>1</sup>, David Knichel<sup>1</sup>, Pascal Sasdrich<sup>1</sup> and Amir Moradi<sup>2</sup>

<sup>1</sup> Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany  
[firstname.lastname@rub.de](mailto:firstname.lastname@rub.de)

<sup>2</sup> University of Cologne, Institute for Computer Science, Cologne, Germany  
[firstname.lastname@uni-koeln.de](mailto:firstname.lastname@uni-koeln.de)

**Abstract.** Accelerated by the increased interconnection of highly accessible devices, the demand for effective and efficient protection of hardware designs against Side-Channel Analysis (SCA) is ever rising, causing its topical relevance to remain immense in both, academia and industry. Among a wide range of proposed countermeasures against SCA, *masking* is a highly promising candidate due to its sound foundations and well-understood security requirements. In addition, formal adversary models have been introduced, aiming to accurately capture real-world attack scenarios while remaining sufficiently simple to efficiently reason about the SCA resilience of designs. Here, the *d*-probing model is the most prominent and well-studied adversary model. Its extension, introduced as the *robust d*-probing model, covers physical defaults occurring in hardware implementations, particularly focusing on *combinational recombinations* (glitches), *memory recombinations* (transitions), and *routing recombinations* (coupling).

With increasing complexity of modern cryptographic designs and logic circuits, formal security verification becomes ever more cumbersome. This started to spark innovative research on automated verification frameworks. Unfortunately, these verification frameworks mostly focus on security verification of hardware circuits in the presence of glitches, but remain limited in identification and verification of transitional leakage. To this end, we extend SILVER, a recently proposed tool for formal security verification of masked logic circuits, to also detect and verify information leakage resulting from combinations of glitches *and* transitions. Based on extensive case studies, we further confirm the accuracy and practical relevance of our methodology when assessing and verifying information leakage in hardware implementations.

**Keywords:** Side-Channel Analysis · Transitional Leakage · Masking · Hardware

## 1 Introduction

**Physical Attacks and Countermeasures.** Even though *design* of secure cryptographic algorithms is a well-researched topic, secure *implementation* remains a late-breaking challenge. Principally, cryptographic algorithms are designed to withstand adversaries in the *black-box* model, limiting adversarial observations to inputs and outputs. However, practical implementations on physical devices usually entail side-channel information. Within the context of such a *gray-box* model, adversaries may observe any physical behavior and characteristics of an electronic device during execution of security-critical applications, such as temporal behavior [Koc96], instantaneous power consumption [KJJ99], Electromagnetic (EM) radiations [GMO01], or temperature and heat dissipation [HS13], in order to reveal secret and sensitive information.

Consequently, many protection mechanisms have been proposed, among which *masking* (based on the concepts of *secret-sharing*) prevails as most promising candidate, due to a formal and sound theoretical foundation [CJRR99]. Specifically in the context of hardware implementations, different masking schemes and flavors have been proposed over the years [ISW03, NRR06, RBN<sup>+</sup>15, GMK17, GM18], continuously improving efficiency and security. At the same time, not a few schemes have been shown to be insecure due to design flaws [MMSS19]. Nowadays, there is a growing awareness that design, implementation, and protection of cryptographic algorithms is still a (mostly) manual, delicate, and error-prone process requiring long-standing expertise and experience in hardware design and security. Moreover, this recognition propels an entirely new branch of research intending to consolidate security models and formal methods to assess and verify the security of hardware implementations.

**Formal Verification of Hardware Security.** Strong theoretical foundations, such as formal models for adversaries and physical execution environments, along with formal verification methods can support and accelerate design, implementation, and verification of secure cryptographic devices and applications. In connection with *hardware masking*, the simple and abstract Ishai, Sahai, and Wagner (ISW)  $d$ -probing adversary and security model [ISW03] usually provides a baseline verification model to reason about security in a *gray-box* context. In its basic appearance, the  $d$ -probing model allows an adversary to arbitrarily observe up to  $d$  intermediate values of an ideal circuit<sup>1</sup> during the processing of sensitive information. Along with some basic assumptions on noise, independence of inputs, and circuit encoding, the security of masked cryptographic implementations can be verified and proved under the  $d$ -probing model.

With respect to physical hardware and digital logic circuits, however, unconsidered and unintentional physical effects, such as *glitches* [MPG05, MS06], *transitions* [CGP<sup>+</sup>12, BGG<sup>+</sup>14], or *couplings* [CBG<sup>+</sup>17], and implementation defects due to architectural conditions, such as *parallelism* [BDF<sup>+</sup>17] or *pipelining* [CGD18], have been shown to be responsible for security degradation of theoretically secure designs and implementations. Most of all, *glitches* are long-known causes for leakage of sensitive information even in the presence of appropriate masking countermeasures [MPG05]. Adjusting the deficiencies in the original ISW  $d$ -probing model, a more robust extension of the standard model has been proposed recently [FGP<sup>+</sup>18], particularly allowing to incorporate unintentional physical defaults as part of the adversarial model.

**Automated Tools for Formal Verification.** Through the ever increasing integration of modern circuits and systems, complexity of hardware designs and implementations grows further. In a sense, this continuous progress and increasing complexity of hardware structures limits manual verification of masked circuits mostly to atomic parts and minor components only, often considered as masked *gadgets* in recent literature. To this end, the development of automated tools assisting designers in formal verification of masked circuits is a natural and long overdue step.

Set within the context of automated formal verification, `maskVerif` [BBD<sup>+</sup>15, BBD<sup>+</sup>16, BBC<sup>+</sup>19] is the first proposal addressing this direction of research. Originally, `maskVerif` was designed to verify  $d$ -probing security for ideal circuits in the presence of transitional leakage, but later versions of the tool have been extended to cover  $d$ -probing security and security notions for *composability*, i.e., Non-Interference (NI) [BBD<sup>+</sup>15] and Strong Non-Interference (SNI) [BBD<sup>+</sup>16], in the presence of glitches, however, still considering transitions for ideal circuits only. While `maskVerif` pursues a language-based formal verification approach, REBECCA [BGI<sup>+</sup>18] and COCO [GHP<sup>+</sup>21] rely on Fourier coefficient

<sup>1</sup>In an ideal circuit, it is supposed that the gates have no delay to propagate the input changes to the output, hence no glitches.

estimation to automatically verify security of masked circuits. Even though REBECCA was the first tool to explicitly consider glitches as physical defaults, it is still limited to verification of  $d$ -probing security only, without supporting verification of additional composability notions (the same holds for COCO, extending REBECCA to verification of masked software implementations on CPUs). Only recently, SILVER [KSM20] has been presented, allowing to verify pure  $d$ -probing security and all recent security notions for composability (NI, SNI, and PINI [CS20]) in the presence of glitches while using an exact verification approach based on Reduced Ordered Binary Decision Diagrams (ROBDDs), avoiding false negatives (in contrast to `maskVerif` which relies on an optimistic approach).

**(In-)complete Modeling of Physical Defaults.** Comparing these existing approaches and tools for formal verification of masked hardware circuits, what becomes immediately apparent is the fact that all tools only consider glitches as undesired physical defaults while neglecting information leakage due to *transitions*. In a sense, even though transitional leakage is well-known to cause security degradation [CGP<sup>+</sup>12, BGG<sup>+</sup>14, CS21], only `maskVerif` implements a rudimentary verification under such conditions, however, focusing on ideal logic circuits only, hence, neglecting glitches as further source of unintentional security degradation. As a consequence, none of the existing tools supports a complete modeling of unintentional physical defaults occurring in hardware, i.e., glitches *and* transitions, for the security verification of masked circuits. Even worse, under certain circumstances, all tools may lead to false positives, i.e., reporting a hardware design secure in the presence of glitches, while the presence of transition leakage actually breaks the security of the masking scheme.

**Our Contributions.** In this work, we translate the transition-related findings of [FGP<sup>+</sup>18] into an algorithmic evaluation approach well suited for its integration into leakage verification tools. The result is a novel probe-extension procedure allowing us to model information leakage due to physical effects originating in combinational and memory recombinations, i.e., glitches and transitions. In a next step, we extend the state-of-the-art leakage verification framework SILVER to assess the security of masked hardware circuits considering glitches and transitions simultaneously. For this, we further extend the capabilities of SILVER to process and analyze iterative digital logic circuits, i.e., digital logic that processes data in a sequential fashion. Eventually, demonstrating the power of our extended model and verification framework as well as the practical relevance of accurately modeling glitches and transitions during security verification, we analyze different iterative 8-bit S-box constructions, proposed by Boss et al. [BGG<sup>+</sup>16], and report information leakage due to transitions, both in formal security verification as well as in experimental Side-Channel Analysis (SCA) evaluations. In a second set of case studies, we deal with iterative circuits, while Hardware Private Circuit (HPC) gadgets are integrated. In contrast to the original version of SILVER, we are able to evaluate such circuits made by Output-Probe-Isolating Non-Interference (O-PINI) secure gadgets as presented in [CS21]. Thanks to our extended version of SILVER, while confirming the issues reported and claims made in [CS21], we provide other insights through theoretical and experimental analyses when HPC gadgets are processed iteratively.

## 2 Background

### 2.1 Notations

We denote Boolean variables  $x \in \mathbb{F}_2$  by lower case letters and vectors of multiple Boolean variables  $X \in \mathbb{F}_2^n$  by upper case letters. To denote single elements of a vector  $X \in \mathbb{F}_2^n$  we use subscripts. Hence,  $x_i \in \mathbb{F}_2$  denotes the element at position  $i$  of  $X$ . Moreover, we use

subscripts and curly brackets to denote a primary input at a specific point in time. For example,  $x_{\{i\}}$  is forwarded to a circuit in clock cycle  $i$ . Further, we denote shares of a variable with superscripts. Hence,  $X^i \in \mathbb{F}_2^n$  denotes share  $i$  of the unshared variable  $X$ . To formalize the probing model, we use upper case bold letters to denote a set of probes, e.g.,  $\mathbf{P}$  while we apply lower case bold letters to denote a single probe  $\mathbf{p} \in \mathbf{P}$ . Again, we denote the  $i^{\text{th}}$  probe  $\mathbf{p}_i \in \mathbf{P}$  of a set with subscripts. For Boolean functions  $f$  and vectorial Boolean functions  $F$  we use sans-serif fonts.

## 2.2 $d$ -Probing Model

In the standard  $d$ -probing model, originally introduced in [ISW03], an adversary is granted the ability to probe up to  $d$  wires in an ideal circuit. This means, every probe is instantaneous and independent of all other probes while providing access to the exact, noise-free, and stable signal of a wire upon invocation of the circuit, i.e., for specific input assignments and states of memory elements. Following this model, a circuit provides  $d^{\text{th}}$ -order probing security, if and only if an adversary is not able to learn anything about the secret, i.e., the joint distribution over observations made by up to  $d$  probes is independent of the distribution of any secret processed by the circuit.

By introducing the *robust* probing model in [FGP<sup>+</sup>18], Faust et al. extended the standard probing model in order to sufficiently capture information leakage originating from physical defaults occurring in physical logic circuits and hardware implementations. More precisely, the work covers *combinational recombinations* (glitches), *memory recombinations* (transitions), and *routing recombinations* (couplings), aiming to accurately cover all phenomena relevant to side-channel leakage in hardware. For each considered physical default, a probe extension was introduced to increase the number of observed values within a circuit according to the considered physical default.

**Glitch-Extended Probes.** *Glitches* are unintentional and undesired signal transitions due to different delay paths and switching delays within a combinational circuit. In order to model such a behavior within the framework of the robust  $d$ -probing adversarial model, any glitch-extended probe allows an adversary to learn all stable inputs (and hence all possible combinations) flowing into the computation of the probed wire. As such, assuming a worst-case scenario, a single glitch-extended probe on a combinational circuit is then substituted by all relevant (standard) probes on the outputs of the last register stage or on primary inputs.

**Transition-Extended Probes.** *Transitions* are unintentional and undesired recombinations of memory contents due to consecutive invocations (e.g., clock cycles) of a circuit. In order to model this behavior within the context of the robust  $d$ -probing adversarial model, each *transition-extended* probe, placed on a memory element, is substituted by two (standard) probes, one on the memory input and one on the memory output, effectively doubling the number of adversarial probes.

**Coupling-Extended Probes.** *Couplings* are unintentional and undesired recombinations of values carried on neighboring wires. In order to model such a behavior in the robust  $d$ -probing adversarial model, each *coupling-extended probe* is replaced by the set of (standard) probes observing the set of neighboring wires.

**$(g, t, c)$ -Extended Probes.** We apply the notation proposed in [FGP<sup>+</sup>18], denoting a  $(g, t, c)$ -extended probe as a glitch-extended (if  $g = 1$ ), transition-extended (if  $t = 1$ ), and coupling-extended (if  $c = 1$ ) probe within the robust  $d$ -probing adversarial model. More specifically, throughout the given work, we will focus on  $(g, t, 0)$ -extended probes, as we intend to analyze circuits on a gate-level for which layout and routing information usually is not available and coupling cannot be modeled with adequate precision.



## 2.3 Boolean Masking

Following the approach of *secret sharing*, Boolean masking splits a secret  $X \in \mathbb{F}_2^n$  into  $s$  independently and uniformly distributed shares  $X^i$ ,  $0 \leq i < s$ , such that the sum over all these shares equals  $X$ , i.e.,  $X = \bigoplus_{i=0}^{s-1} X^i$ . Known as uniform sharing, this is commonly achieved by drawing  $X^i$  uniformly and at random from  $\mathbb{F}_2^n$  for all  $0 \leq i < s - 1$  and computing the remaining share as  $X^{s-1} = X \oplus \bigoplus_{i=0}^{s-2} X^i$ , transforming the unshared secret  $X \in \mathbb{F}_2^n$  into its shared representation  $X' = (X^i)_{0 \leq i < s} \in \mathbb{F}_2^{n \times s}$ . Naturally, this approach requires splitting any secret in at least  $d + 1$  shares to achieve security against an adversary that is able to probe up to  $d$  wires in a circuit, i.e., to achieve  $d$ -probing security.

## 2.4 Threshold Implementation

Threshold Implementations (TIs) have been introduced and developed as a way to realize masked circuits and to avoid leakage in hardware implementations caused due to glitches [Bil15, CBR<sup>+</sup>15, BGN<sup>+</sup>14, NRR06, NRS08]. The number of input ( $s_{in}$ ) and output sharings ( $s_{out}$ ) necessary to realize a  $d^{th}$ -order TI of a given Boolean function  $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$  with algebraic degree  $t$  is restricted by  $s_{in} \geq t \times d + 1$  and  $s_{out} \geq \binom{s_{in}}{t}$  where  $F_{TI} : \mathbb{F}_2^{n \times s_{in}} \mapsto \mathbb{F}_2^{m \times s_{out}}$  denotes the resulting TI which can be denoted as a set of  $s_{out}$  component functions  $F_{TI}^{i \leq s_{out}} : \mathbb{F}_2^{n \times s_{in}} \mapsto \mathbb{F}_2^m$ . A secure  $d^{th}$ -order TI further should fulfill the following properties:

**Correctness.** Let  $X' \in \mathbb{F}_2^{n \times s_{in}}$  be a valid Boolean masking of  $X \in \mathbb{F}_2^n$ . Then  $Y' = F_{TI}(X') \in \mathbb{F}_2^{m \times s_{out}}$  is a valid Boolean masking of  $Y = F(X) \in \mathbb{F}_2^m$ .

**Non-Completeness.** Any combination of up to  $d$  component functions  $F_{TI}^i$  of  $F_{TI}$  must be independent of at least one input share for every secret. Intuitively, when solitary considering a circuit fulfilling non-completeness, an adversary in the  $d$ -probing model will not learn anything about a secret  $X$ , as non-completeness guarantees that any observation made by up to  $d$  probes will be independent of at least one circuit share and due to the uniformity property of Boolean masking, these circuit shares will hence be independent of the secret.

**Uniformity.** When grouped corresponding to their unshared input value  $X$ , each valid output sharing  $Y'$  of  $Y = F(X)$  within this group has to occur  $p$  times, where  $p$  is constant over all  $X$  and  $Y'$ , while each invalid sharing does not occur at all. This guarantees that  $Y'$  always is a uniform sharing of  $Y$  when given as input to a subsequent circuit.

## 2.5 Trivially Composable Gadgets

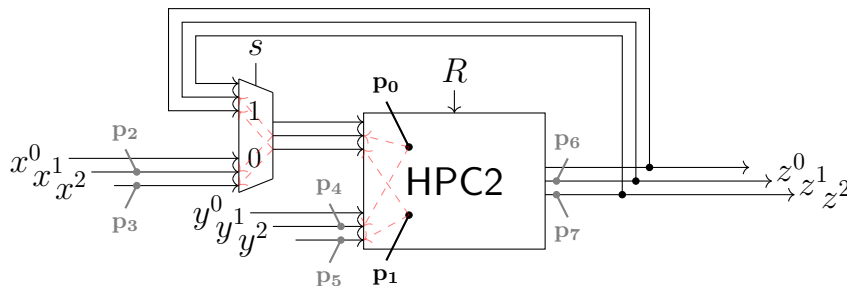
Finding secure, masked circuits for high security orders and large functions has proven to be a hard task. For this, a wide variety of composability notions have been established, aiming to define properties of masked circuits that are sufficient to guarantee security in the  $d$ -probing model when combined to a larger design. This enables the construction of small sub-circuits (typically realizing simple two-input gates for AND and XOR) that lead to  $d$ -probing secure circuits when combined and interconnected. In recent literature, these composable, atomic sub-circuits are commonly referred to as *gadgets*.

In the course of this direction of research, Non-Interference (NI) and Strong Non-Interference (SNI) [BBD<sup>+</sup>15, BBD<sup>+</sup>16] were introduced first, where SNI remedied composability flaws discovered for NI by further restricting *propagation* [CS20] of probes placed on output wires of a gadget. As the scope of SNI was initially limited to single-output gadgets, Cassiers et al. [CS20] extended this notion to capture multiple-output gadgets as well, but

in the same work, they introduced the notion of Probe-Isolating Non-Interference (PINI) as a way to further reduce overhead requirements of composable gadgets and allowing trivial implementation of linear functions.

**Probe-Isolating Non-Interference.** Following the principles of Domain Oriented Masking (DOM) [GMK17], share domains were introduced for PINI [CS20] and output probes are restricted to propagate within their own share domain while internal probes are limited to propagate into a single, but arbitrary, domain. This allows trivial implementation of linear functions, i.e., their share-wise application, while enabling trivial composition of gadgets fulfilling the notion of PINI by simply interconnecting different input- and output-shares with respect to their share domain, i.e., not mixing share domains through interconnections.

**Output-Probe-Isolating Non-Interferences.** Recently, in order to model and achieve trivial composition under transitions, the notion of O-PINI was introduced in [CS21], as an extension to the original definition of PINI, aiming to solve issues that may occur when there exist a feedback loop in the design, e.g., one input of a gadget depends on an output of itself. In order to model this, at first, it is determined in which input share domains the original probes propagate, before output probes are added (if they do not already exist) to all these domains. The gadgets must then be PINI with respect to the new set of probes in its original definition. This extension is motivated by the fact that adding an internal probe placed on a gadget allows an additional – but arbitrary – domain to be in the set of input shares resulting from propagation. Now, if there is a feedback loop in the design and an output of the gadget is input to itself, these input shares will propagate into the previous iteration, which can be effectively modeled by putting another probe on the corresponding output domain, possibly reducing the security order. For example, as presented in [CS21], Figure 1 visualizes why HPC2 is not O-PINI when initialized for the second security order. For three input shares, we consider  $\mathbf{p}_0$  and  $\mathbf{p}_1$  placed on the gadget internals  $x^1 r_{01}$  propagating into share domain 1 (due to  $x^1$ ) and  $x^2 r_{21}$  propagating into share domain 2 (due to  $x^2$ ). Since there exists a feedback loop from its outputs to the gadget’s inputs,  $\mathbf{p}_0$  and  $\mathbf{p}_1$  would further propagate into the outputs belonging to share domain 1 and 2, which is similar to placing additional output probes  $\mathbf{p}_6$  and  $\mathbf{p}_7$ . Now, given  $\{\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_6, \mathbf{p}_7\}$ , i.e., two output probes placed on domain 1 and domain 2 and two internal probes, PINI only guarantees that in this case the propagation is limited to input share domain 1 and 2 (due to  $\mathbf{p}_6$  and  $\mathbf{p}_7$ ) and at most two other, domains (due to the internal probes) which may potentially include share domain 0. In summary,  $\{\mathbf{p}_0, \mathbf{p}_1\}$  may now propagate into three share domains 0, 1, and 2, rendering the design insecure.



**Figure 1:** Iterative HPC2 gadget.

By adding output probes corresponding to the input share domains resulting from propagation and still guaranteeing that the original number of input domains, in which the new set of probes propagates into, does not increase, O-PINI enables trivial composition of iterative designs in the presence of transitions.

**HPC Gadgets.** In [CGLS21], Cassiers et al. introduced novel gadget constructions called Hardware Private Circuits (HPCs). This included HPC2, a gadget realizing a 2-input AND gate and offering trivial composability under the notion of PINI in the glitch-extended probing model for arbitrary security orders. This allows trivial construction of secure circuits by simply substituting AND gates with its shared representation, i.e., HPC gadgets. The security and composability guarantees provided by these gadgets are in conformity with the notion of PINI in the presence of only glitches, excluding transitions. As a refined notion which is able to handle transitions, O-PINI was introduced in [CS21], and at the same time HPC2 was further adapted to be in conformity with it and hence be trivially composable under transitions combined with glitches.

## 2.6 Formal Verification

In order to verify the security and composability of a design in the  $d$ -probing model, a wide variety of tools have been proposed which differ in their abstraction and accuracy levels [BGI<sup>+</sup>18, BBC<sup>+</sup>19, KSM20, BDM<sup>+</sup>20, BBD<sup>+</sup>16, BGR18, BBD<sup>+</sup>15, CGLS21]. However, all these tools offer a valuable assistance in finding provable secure masking schemes and enable a designer to catch flaws in an early stage of the design process.

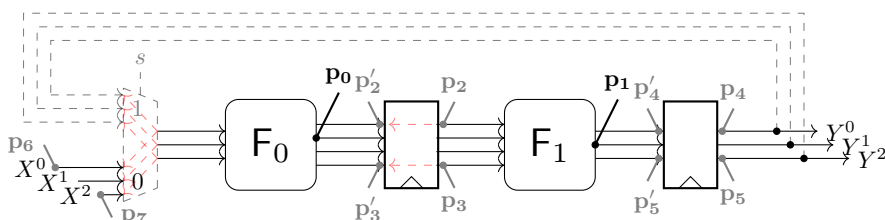
**SILVER.** In [KSM20], Knichel et al. introduced SILVER, a verification tool that utilizes a methodology based on Binary Decision Diagrams (BDDs) for checking statistical independence between observations made by (glitch-extended) probes and secret values or set of shares, effectively enabling a verification of all common security and composability notions (including PINI) in the standard and glitch-extended robust  $d$ -probing model. As SILVER currently is the only publicly-available<sup>2</sup> tool supporting the notion of PINI and is free of false negatives, we opted to integrate our results into its verification framework. Fortunately, following the concepts outlined Section 2.2, extending the probes with respect to physical defaults is a simple pre-processing step, possibly increasing the number of (standard) probes which have to be considered. This means, the core of SILVER, i.e., the check for statistical independence utilizing BDDs, remains unchanged regardless of any probe extension (which is already demonstrated for glitch-extended probes in the current version of the tool).

**Limitations.** As SILVER constructs BDDs for representing Boolean functions, the verification run time mainly depends on the complexity of the underlying Boolean functions, in particular, the number of product terms in the disjunctive normal form of the circuit [KSM20]. Therefore, SILVER is applicable to small circuits like gadgets or single S-Boxes, but not to a full cipher design. Moreover, the complexity of constructing all relevant probe combinations grows exponentially with the security order, and each combination itself encompasses a larger set of random variables. Consequently, higher-order leakage verification with SILVER is only possible for small circuits. Further, due to its nature, the construction of BDDs for a circuit containing a loop is not possible. Hence, SILVER is not able to handle such circuits.

## 3 Transitional Leakage

As discussed in Section 2.2, the  $(g, t, c)$ -robust  $d$ -probing security model formalizes models for capturing three different type of physical defaults (glitches, transitions, and couplings). In particular, modeling of any considered default [FGP<sup>+</sup>18] or even a combination of multiple defaults relies on extending a  $d$ -set of probes on intermediate values according to the corresponding extension scheme.

<sup>2</sup><https://github.com/Chair-for-Security-Engineering/SILVER>



**Figure 2:** Iterative design of  $F$  with static primary inputs.

Currently, verifications are mostly performed for the widely used specification of  $(1, 0, 0)$ -robust  $d$ -probing, i.e., a  $d$ -set, generated by glitch-extended probes, is used to take combinational recombinations into account. However, to further allow the verification given the combined occurrence of glitches and transitions, we present an automated procedure for the combination of glitch-extended and transition-extended probes. For this, the final set of probes can then be used for verification within the  $(1, 1, 0)$ -robust  $d$ -probing model as part of security verification tools such as SILVER [KSM20].

For the sake of clarity, we start this section by presenting two examples and explain how the expansion of probes is performed when (1) the primary inputs are stable and (2) they change during the evaluation of the circuit. As discussed in [CS21], leakages due to transitions often lead to problems if the same physical instance of a function  $F$  is evaluated multiple times by different but not necessarily independent inputs. This holds for the evaluation of an *iterative circuit* which is the underlying architecture of our experiments. For this, we first define the structure of an iterative circuit.

**Definition 1** (Iterative Circuit). An *iterative circuit* implements an *iteration function*  $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n, U \mapsto Y$  driven by a multiplexer  $\text{MUX} : \mathbb{F}_2^n \times \mathbb{F}_2^n \times \mathbb{F}_2 \rightarrow \mathbb{F}_2^n, (X, V, s) \mapsto U$  as

$$U = \begin{cases} X & \text{if } s = 0, \\ V & \text{otherwise} \end{cases},$$

while  $X$  denotes the primary input and  $Y$  the primary output. Further,  $V \subseteq Y$  stands for the feedback meaning that it is taken from the output of  $F$ . One *execution* of the circuit means iterating  $F$   $k$  times, for a constant  $k \geq 2$ . Depending on  $s$ , the MUX selects either the primary input  $X$  or the feedback  $V$  to be given to  $F$ . In a typical case,  $X$  is selected for the first iteration while a part of the output of  $F$  is given to its input for the following iterations.

Given this definition, we consider an iterative circuit for the examples given below, which realize an exemplary function  $F = F_1 \circ F_0$  using two combinational circuits  $F_0$  and  $F_1$  whose outputs are directly stored in registers. Hence, each iteration of  $F$  takes two clock cycles. Further, the circuit receives three shares of an  $n$ -bit primary input  $(X^0, X^1, X^2)$ ,  $X^i \in \mathbb{F}_2^n$ , while the shared output of the function is also fed back as the new input, selected by a multiplexer. After  $k$  iterations, the circuit output is taken as the (shared) result of such an execution  $(Y^0, Y^1, Y^2)$ ,  $Y^i \in \mathbb{F}_2^n$ . Further, in order to ensure the correctness of the operation, the select signal of the multiplexer  $s = 0$  if  $k = 0$  (first iteration) while  $s = 1$  for any subsequent iteration  $k \neq 0$ . Figure 2 and Figure 3 depict such an iterative circuit.

**Example 1** (Static Primary Inputs). For the first example, we suppose that all primary inputs remain stable throughout the entire execution. In this sense, the primary inputs do not change during each of the  $k$  iterations. Now, let us consider a set of two probes  $\mathbf{P} = \{\mathbf{p}_0, \mathbf{p}_1\}$  placed on an output of  $F_0$  and  $F_1$  respectively, as shown in Figure 2. In the following, we intend to extend the set  $\mathbf{P}$  to cover both glitches *and* transitions.

**Step 1: Glitch-Extension.** We start to model glitches for all  $\mathbf{p} \in \mathbf{P}$  by adding the relevant probes, according to Section 2.2, into  $\mathbf{P}_g$ , i.e., the set of probes after glitch-extension. As  $\mathbf{p}_0$  (resp.  $\mathbf{p}_1$ ) is placed on one output share of  $F_0$  (resp.  $F_1$ ), we need to add probes on all inputs of the combinational logic that contribute to the computation of  $\mathbf{p}_0$  (resp.  $\mathbf{p}_1$ ). As an exemplary case, we assume the following extensions.

$$\mathbf{p}_1 \xrightarrow{(1,0,0)} \{\mathbf{p}_2, \mathbf{p}_3\}, \quad \mathbf{p}_0 \xrightarrow{(1,0,0)} \{\mathbf{p}_4, \mathbf{p}_5, \mathbf{p}_6, \mathbf{p}_7\}, \quad \mathbf{P}_g = \{\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5, \mathbf{p}_6, \mathbf{p}_7\}$$

Supposing that  $\mathbf{p}_0$  is extended to two input shares of  $F_0$ , we need to examine all corresponding inputs of the multiplexer. More precisely, if  $s$  switches from 0 to 1, the multiplexer output replaces the primary inputs with the feedback signals. Hence, any glitches, propagated to the multiplexer outputs, depend on the primary inputs as well as the feedback signals, i.e., the outputs of the previously accomplished iteration. We model this effect by adding probes on the corresponding feedback signals to  $\mathbf{P}_g$ , i.e., adding  $\mathbf{p}_4$  and  $\mathbf{p}_5$ , and on the corresponding primary inputs, i.e.,  $\mathbf{p}_6$  and  $\mathbf{p}_7$ .

**Step 2: Transition-Extension.** Second, we now extend the probes in  $\mathbf{P}_g$  to take transitions into account. Note, that it is also possible to first extend the probes based on transitions and then continue with the extension scheme for glitches. This eventually will result in the same  $d$ -set of probes that can be evaluated for statistical independence.

In a sense, according to [FGP<sup>+</sup>18], transitions combine two values that are consecutively seen on the same register. Modeling the overwriting effect for registers allows the extension of a probe  $\mathbf{p} \in \mathbf{P}_g$  placed on a register output by another probe  $\mathbf{p}'$  on the corresponding register input. Consequently, our approach analyzes  $\mathbf{P}_g$  and extends all contained probes after the glitch-extension that are already at the register outputs (or primary inputs). For instance, as  $\mathbf{p}_2$  (resp.  $\mathbf{p}_3$ ) is placed on an output of the first register stage, it holds that:

$$\mathbf{p}_2 \xrightarrow{(0,1,0)} \{\mathbf{p}_2, \mathbf{p}'_2\}, \quad \mathbf{p}_3 \xrightarrow{(0,1,0)} \{\mathbf{p}_3, \mathbf{p}'_3\}$$

The same extensions must be applied on  $\mathbf{p}_4$  and  $\mathbf{p}_5$  placed on outputs of the second register stage. Hence, it holds that:

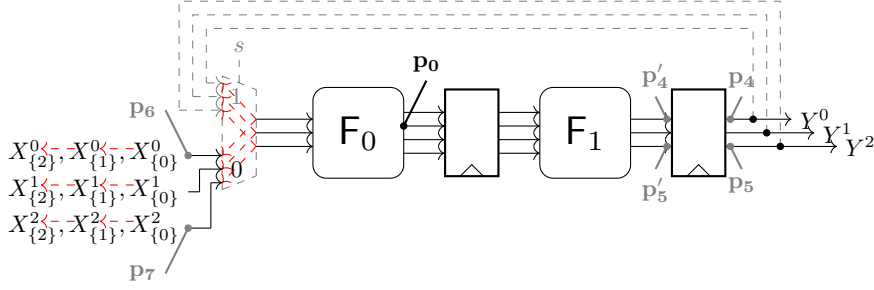
$$\mathbf{p}_4 \xrightarrow{(0,1,0)} \{\mathbf{p}_4, \mathbf{p}'_4\}, \quad \mathbf{p}_5 \xrightarrow{(0,1,0)} \{\mathbf{p}_5, \mathbf{p}'_5\}, \\ \mathbf{P}_{g,t} = \{\mathbf{p}_2, \mathbf{p}'_2, \mathbf{p}_3, \mathbf{p}'_3, \mathbf{p}_4, \mathbf{p}'_4, \mathbf{p}_5, \mathbf{p}'_5, \mathbf{p}_6, \mathbf{p}_7\}.$$

As a consequence,  $\{\mathbf{p}'_2, \mathbf{p}'_3, \mathbf{p}'_4, \mathbf{p}'_5\}$  denote the transition-extension probes on the register inputs. Note that all register inputs are stable when stored. Hence, no further glitch-extension on  $\{\mathbf{p}'_2, \mathbf{p}'_3, \mathbf{p}'_4, \mathbf{p}'_5\}$  is necessary and this step of our algorithm terminates here. Ultimately,  $\mathbf{P}_{g,t}$  now covers glitches and transitions, and its statistical independence to the secrets (i.e., the unmasked primary input  $X = (X^0 \oplus X^1 \oplus X^2)$ ) can be examined implying the evaluation of two probes  $\{\mathbf{p}_0, \mathbf{p}_1\}$  under a  $(1, 1, 0)$ -robust 2-probing model.

### 3.1 Changes on Primary Inputs

In the previous example, we mainly followed the concepts and assumptions provided in [CS21], particularly assuming that all primary inputs remain stable and static for the entire execution of the circuit through multiple iterations. In one of the case studies, which we present in Section 5, we demonstrate that such an assumption in fact can be the source of security flaws. For this, in the following example, we allow transitions on primary inputs such that each primary input can change at every clock cycle.

**Example 2** (Dynamic Primary Inputs). In order to deal with inputs being changed at every clock cycle, we introduce the notion of *input sequences*, denoting input values



**Figure 3:** Iterative design of  $F$  with transitional primary inputs.

given per clock cycle. Hence, with such an implicit notion of time, we define the primary input  $X$  at cycle  $l$  as  $X_{\{l\}}$ . We further consider the same iterative circuit as shown in Example 1, but, for the sake of simplicity, we limit the set of probes to a single probe  $\mathbf{P} = \{\mathbf{p}_0\}$  as visualized in Figure 3. Again, we start with extending the probes based on glitches. Following the same procedure as in Example 1,  $\mathbf{p}_0$  is extended such that  $\mathbf{p}_0 \xrightarrow{(1,0,0)} \{\mathbf{p}_4, \mathbf{p}_5, \mathbf{p}_6, \mathbf{p}_7\}$ . Now, the fundamental difference to Example 1 is that we have different input values consecutively given in every clock cycle. As a consequence, transitions not only occur between input and output of registers, but may also occur due to the consecutive primary input values. Hence, modeling the transitions by extending the probes to cover registers, i.e.,  $\{\mathbf{p}'_4, \mathbf{p}'_5\}$ , is not sufficient. For this, let us examine the probes on the primary inputs  $\{\mathbf{p}_6, \mathbf{p}_7\}$ . Generally, if we are in clock cycle  $l$ , probe  $\mathbf{p}_6$  observes a bit of  $X_{\{l\}}^0$  and  $\mathbf{p}_7$  a bit of  $X_{\{l\}}^2$ . Obviously, such an extension does not consider the fact that input values in cycle  $l$  have been replaced by the associated values of cycle  $l + 1$ , i.e., the corresponding bits of  $X_{\{l+1\}}^0$  and  $X_{\{l+1\}}^2$ . Therefore, we need to include two additional standard probes. Since the feedback signals and primary inputs have two clock cycles distance (see Figure 3), if we are in the second clock cycle ( $l = 1$ ), the first iteration of the circuit  $k = 1$  is accomplished, the feedback signals traverse the output of  $F_1$  being stored in the register, and the select signal of the multiplexer switches. This means that  $\mathbf{p}_4$  and  $\mathbf{p}_5$  observe the feedback value, and  $\mathbf{p}_6$  and  $\mathbf{p}_7$  the corresponding bits of  $X_{\{1\}}^0$  and  $X_{\{1\}}^2$ . Due to the primary input transitions,  $\mathbf{p}_6$  and  $\mathbf{p}_7$  are also extended to  $\mathbf{p}'_6$  and  $\mathbf{p}'_7$  observing the corresponding bits of  $X_{\{2\}}^0$  and  $X_{\{2\}}^2$ . As a consequence, the resulting extensions can be modeled as:

$$\begin{aligned} \mathbf{p}_4 &\xrightarrow{(0,1,0)} \{\mathbf{p}_4, \mathbf{p}'_4\}, & \mathbf{p}_5 &\xrightarrow{(0,1,0)} \{\mathbf{p}_5, \mathbf{p}'_5\}, & \mathbf{p}_6 &\xrightarrow{(0,1,0)} \{\mathbf{p}_6, \mathbf{p}'_6\}, & \mathbf{p}_7 &\xrightarrow{(0,1,0)} \{\mathbf{p}_7, \mathbf{p}'_7\} \\ \mathbf{P}_{g,t\{2\}} &= \{\mathbf{p}_4, \mathbf{p}'_4, \mathbf{p}_5, \mathbf{p}'_5, \mathbf{p}_6, \mathbf{p}'_6, \mathbf{p}_7, \mathbf{p}'_7\}. \end{aligned}$$

Note that in the former cycles  $l < 1$ , the feedback signal does not yet carry the output of  $F_1$  depending on the given primary inputs. Hence, the extension of the probes is slightly different. For example, in the first clock cycle ( $l = 0$ ),  $\mathbf{p}_0$  extends to

$$\mathbf{p}_0 \xrightarrow{(1,1,0)} \mathbf{P}_{g,t\{1\}} = \{\mathbf{p}_6, \mathbf{p}'_6, \mathbf{p}_7, \mathbf{p}'_7\},$$

and the following primary inputs are probed:

$$\mathbf{p}_6 \xleftarrow{\text{probe}} X_{\{0\}}^0, \quad \mathbf{p}'_6 \xleftarrow{\text{probe}} X_{\{1\}}^0, \quad \mathbf{p}_7 \xleftarrow{\text{probe}} X_{\{0\}}^2, \quad \mathbf{p}'_7 \xleftarrow{\text{probe}} X_{\{1\}}^2$$

### 3.2 Modeling Glitch- and Transition-Extended Probes

After giving a first intuition on the modeling and generation of  $(1, 1, 0)$ -extended probes in the  $d$ -probing model using the two above examples, we now formally define the algorithmic

solution for the identified problems and challenges. In addition, we show how to integrate these algorithmic concepts into state-of-the-art formal verification tools, such as SILVER<sup>3</sup>.

**Circuit Model.** We model a circuit as *Directed Acyclic Graph (DAG)* that represents all gates as nodes and all wires as edges. Hence, as the first step, such a graph should be made for the circuit under evaluation. For circuits without loops, this can be trivially done, knowing the netlist of the circuit.<sup>4</sup> However, as shown above, we deal with iterative circuits, which contain loops. To handle this, and extract a loop-free graph<sup>5</sup>, we remove all loops and store a list of transitions  $\mathcal{F}$  indicating the feedback signals. Each list entry  $f \in \mathcal{F}$  formalizes a single feedback loop together with its corresponding input multiplexer. We denote each  $f \in \mathcal{F}$  as a triple  $f = (i, j, l)$ , where  $i \in \mathcal{I}$  denotes a primary input overwritten by a feedback signal  $j$  (i.e. both inputs of the multiplexer) during clock cycle  $l$ . With  $\mathcal{F}$ , we keep all information regarding loops while structurally removing them from the circuit.  $\mathcal{F}$  itself is created by analyzing all initial multiplexers  $\text{MUX}(i, j, s)$  that select (depending on the selection signal  $s$ ) if a primary input  $i$  or a feedback signal  $j$  is given to the circuit. The corresponding clock cycle can then be found by analyzing the latency of the feedback signal. As  $j$  is computed during one iteration,  $l$  corresponds to the number of clock cycles per iteration. For example, if we consider Figure 2, we analyze and remove three multiplexers  $\text{MUX}_0(X^0, Y^0, s)$ ,  $\text{MUX}_1(X^1, Y^1, s)$ , and  $\text{MUX}_2(X^2, Y^2, s)$ . The list of transitions is  $\mathcal{F} = \{(X^0, Y^0, 2), (X^1, Y^1, 2), (X^2, Y^2, 2)\}$  and  $\mathcal{I} = \{X^0, X^1, X^2\}$  which are given to  $F_1$ . Based on the given circuit model, we formalize the glitch and transition extension scheme as shown in Algorithm 1. For better understanding, we define the following functions:

**gate**( $w$ ): returns the source gate of wire  $w$ .  
**type**( $g$ ): returns **reg** if gate  $g$  is a register.  
**inputs**( $g$ ): returns a set containing all input wires of gate  $g$ .  
**dfs**( $w$ ): performs a depth-first search starting at wire  $w$  but stops going deeper if a found wire  $w'$  carries a synchronized element (either register output or primary input). It returns all found  $w'$ .

Algorithm 1 receives a  $d$ -set of probes together with a manually created list of all primary inputs  $\mathcal{I}$  and a list of all transitions based on feedback signals  $\mathcal{F}$  for all evaluated points in time. The output of Algorithm 1 is the set of probes after glitch and transition extension.

**Modeling Glitches.** For any  $\mathbf{p} \in \mathbf{P}$ , we model glitches by placing probes on all synchronized elements (either register outputs or primary inputs) that contribute to the computation of the value that  $\mathbf{p}$  observes. We realize the extension by a depth-first search starting at  $\mathbf{p}$  as the root node. The search itself is performed backwards and stops evaluating a branch if a synchronized element is found. Only for the synchronized element, we add a probe on its output wire. This can be seen in Lines 6-8 of Algorithm 1. Since the circuit became loop-free, for any probe placed on an primary input, we check if the primary input is input of an initial multiplexer during the evaluated clock cycle. This is done by searching the tuple with the primary input and the evaluation cycle in  $\mathcal{F}$ . If the tuple exists, we place probes on both multiplexer inputs. Note that the glitches related to both primary input and feedback occur during an individual clock cycle and on a specific input given during this clock cycle. Hence, we create one set of probes per clock cycle containing its individual probes. This can be seen in Lines 10-18 of Algorithm 1.

<sup>3</sup>We opted to integrate our solution into SILVER as it allows to avoid false negatives and provides extensive support for additional composability notions.

<sup>4</sup>In digital circuit design, a netlist is a description of the connectivity of a circuit. In its simplest form, a netlist consists of a list of the cells in a circuit and a list of the nodes they are connected to.

<sup>5</sup>It is essential for some verification tools, e.g., SILVER, as evaluating, e.g., constructing BDDs of a circuit with loop, becomes impossible.

**Algorithm 1** Glitch and transition extension of a probing set

---

```

Input:  $\mathbf{P}$  ▷ Probing set  $\mathbf{P}$ 
Input:  $\mathcal{I}, \mathcal{F}$  ▷ List of primary inputs  $\mathcal{I}$ , list of feedback signals  $\mathcal{F}$ 
Input:  $n$  ▷ Total latency of one iteration
Output:  $\mathbf{P}_{g,t\{0\}}, \mathbf{P}_{g,t\{1\}}, \dots, \mathbf{P}_{g,t\{n-1\}}$  ▷ Final probing sets for all clock cycles
1:
2:  $\mathbf{P}_g \leftarrow \emptyset$  ▷ Initialize set of probes
3: for  $\forall l \in \{0, \dots, n-1\}$  do
4:    $\mathbf{R}_{\{l\}} \leftarrow \emptyset, \mathbf{S}_{\{l\}} \leftarrow \emptyset, \mathbf{P}_{g,t\{l\}} \leftarrow \emptyset$  ▷ Initialize set of probes for cycle  $l$ 
5: end for

6: for all  $\mathbf{p} \in \mathbf{P}$  do ▷ Perform glitch-extension
7:    $\mathbf{P}_g \leftarrow \mathbf{P}_g \cup \text{dfs}(\mathbf{p})$ 
8: end for

9: for  $\forall l \in \{0, \dots, n-1\}$  do
10:  for all  $\mathbf{p} \in \mathbf{P}_g$  do ▷ Perform glitch-extension on feedbacks
11:     $x \leftarrow \mathbf{p}$ 
12:    if  $\exists(x, y, l) \in \mathcal{F}$  then ▷ Check if  $x$  exist in  $\mathcal{F}$ 
13:       $\mathbf{p}_1 \xleftarrow{\text{probe}} x, \mathbf{p}_2 \xleftarrow{\text{probe}} y$ 
14:       $\mathbf{R}_{\{l\}} \leftarrow \mathbf{R}_{\{l\}} \cup \{\text{dfs}(\mathbf{p}_1), \text{dfs}(\mathbf{p}_2)\}$ 
15:    else
16:       $\mathbf{R}_{\{l\}} \leftarrow \mathbf{R}_{\{l\}} \cup \{\mathbf{p}\}$ 
17:    end if
18:  end for

19:  for all  $\mathbf{r} \in \mathbf{R}_{\{l\}}$  do ▷ Perform transition-extension on registers
20:    if  $\text{type}(\mathbf{r}) = \text{reg}$  then ▷ Search all probes on register outputs
21:       $\{x\} \leftarrow \text{inputs}(\mathbf{r})$ 
22:      if  $\exists(x, y, l) \in \mathcal{F}$  then
23:         $\mathbf{r}' \xleftarrow{\text{probe}} \{y\}$ 
24:      else
25:         $\mathbf{r}' \xleftarrow{\text{probe}} \{x\}$ 
26:      end if
27:       $\mathbf{S}_{\{l\}} \leftarrow \mathbf{S}_{\{l\}} \cup \{\mathbf{r}, \mathbf{r}'\}$ 
28:    else
29:       $\mathbf{S}_{\{l\}} \leftarrow \mathbf{S}_{\{l\}} \cup \{\mathbf{r}\}$ 
30:    end if
31:  end for

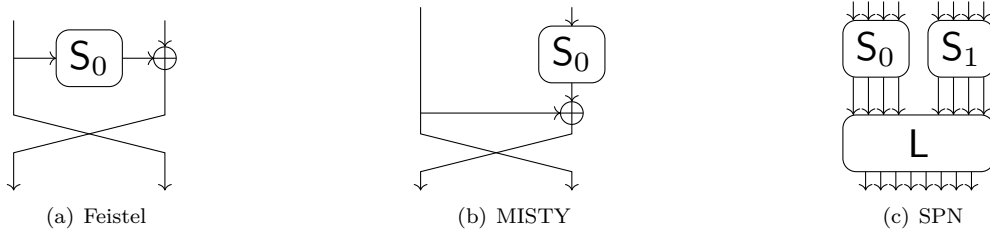
32:  for all  $\mathbf{s} \in \mathbf{S}_{\{l\}}$  do ▷ Perform transition-extension on primary inputs
33:    if  $l > 0 \wedge \mathbf{s} = x_{\{l-1\}} \in \mathcal{I}$  then
34:       $\mathbf{s}' \xleftarrow{\text{probe}} x_{\{l-1\}}$  ▷ Add the primary input of the previous clock cycle
35:       $\mathbf{P}_{g,t\{l\}} \leftarrow \mathbf{P}_{g,t\{l\}} \cup \{\mathbf{s}, \mathbf{s}'\}$ 
36:    else
37:       $\mathbf{P}_{g,t\{l\}} \leftarrow \mathbf{P}_{g,t\{l\}} \cup \{\mathbf{s}\}$ 
38:    end if
39:  end for
40: end for

```

---

**Modeling Transitions.** For any probe  $p \in \mathbf{P}_g$  placed on a register output, we model transitions on the corresponding register by placing a standard probe on the register input.





**Figure 4:** The underlying constructions of the 8-bit S-boxes introduced in [BGG<sup>+</sup>17].

Technically, we place the probe on the wire that drives the register. This can be seen in Lines 19-31 of Algorithm 1. For any probe placed on a primary input  $x_{\{l\}}$  we place an additional standard probe on  $x_{\{l-1\}}$ . This step must be repeated for all evaluated clock cycles. This can be seen in Lines 32-39 of Algorithm 1.

### 3.3 Integration into SILVER

Since manual verification is time-consuming and hardly feasible for larger constructions, we opted to automate our approach and integrated our probe-extension algorithm into the already existing leakage verification tool SILVER [KSM20]. Given that SILVER operates on an annotated gate-level netlist, we extended the initial parsing and pre-processing procedures to support the conversion of an iterative circuit to a loop-free graph as well as the generation of the list of transitions  $\mathcal{F}$ . Further, transitions due to sequences of primary inputs are indicated through an extended annotation scheme in the original gate-level netlist.

In terms of analysis and verification, since SILVER already implements the security verification within the  $(1, 0, 0)$ -robust probing model, we leave the already provided glitch-extension scheme unchanged but only integrate our novel transition-extension scheme accordingly, as shown above. Eventually, the final statistical independence check is now performed on the newly created set of probes  $\mathbf{P}_{g,t\{l\}}$  instead of  $\mathbf{P}$  or  $\mathbf{P}_g$ , as before. As a result, if SILVER detects a statistical dependency for any  $\mathbf{P}_{g,t\{l\}}$ , i.e., the set of probes during cycle  $l$ , it will report information leakage accordingly.

## 4 Case Study 1: Strong 8-bit S-boxes

In order to apply the above-presented leakage evaluation procedure based on iterative circuits existing in literature, we consider the strong 8-bit S-box designs originally proposed at CHES 2016 [BGG<sup>+</sup>16] and later extended in [BGG<sup>+</sup>17]. During the following case study, we evaluate all S-box designs with our extended version of SILVER and compare the outcome with practical results based on physical measurements.

**S-Box Architecture.** The core idea of [BGG<sup>+</sup>17] is to construct a set of 8-bit S-boxes using smaller 4-bit S-boxes and some linear operations in known constructions, namely a Feistel network, a MISTY construction, or a Substitution Permutation Network (SPN). Figure 4 presents all such constructions, where the 4-bit S-boxes are denoted by  $S_i$  and a linear layer by  $L$ . These constructions should be iterated  $k$  times with  $k \geq 2$  to build a cryptographically strong 8-bit S-box. This allowed the authors to build a relatively small masked circuit (TI) trivially realizing one round of such constructions (denoted by SB) and iterate it to achieve an area-efficient masked implementation of the 8-bit S-box. Figure 5 shows such an iterative design, which is similar to what the authors proposed in [BGG<sup>+</sup>17, Fig. 3(c)]. The authors have mainly constructed first-order secure TIs of their designs using

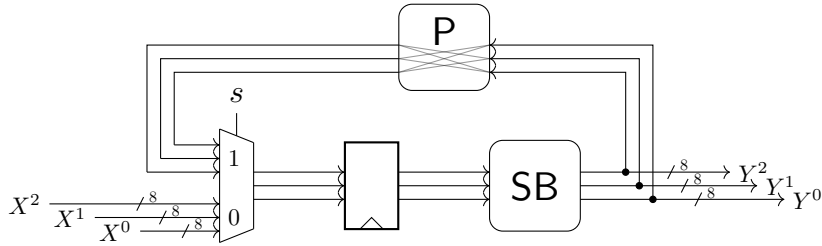


Figure 5: Iterative S-box architecture.

$td + 1 = 3$  shares, which satisfy  $(1, 0, 0)$ -robust 1-probing security. As shown in Figure 5, all input shares of SB are synchronized by a single register stage. Consequently, each iteration of SB is performed within a single clock cycle. In total, the authors introduced a set of eight different S-boxes  $SB_1$  to  $SB_8$ . We give the basic properties of each S-box in Table 1, where Perm. refers to Figure 4(c) with linear layer  $L$  being a bit permutation, and Matrix a matrix multiplication.

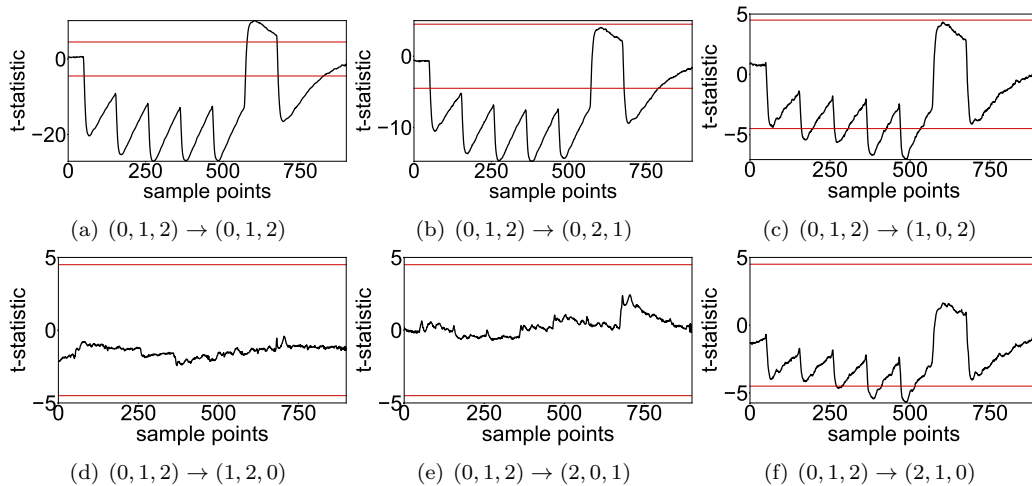
We implemented the same circuits receiving three input shares  $(X^0, X^1, X^2) \in \mathbb{F}_2^{8 \times 3}$  and returning three output shares  $(Y^0, Y^1, Y^2) \in \mathbb{F}_2^{8 \times 3}$ . The output of the combinational logic builds the feedback signal and is given as the new input to the circuit if  $k \neq 0$ . This is handled by a multiplexer driven by the select signal  $s \in \mathbb{F}_2$ . Additionally, we have instantiated a share permutation module, denoted by  $P$  in the feedback path (see Figure 5). The purpose of such a module is to permute the shares. Although it has not been discussed or pointed out in [BGG<sup>+</sup>17], the output shares of SB should not necessarily be identically given to its input shares in the successive iteration. We denote this permutation as a mapping of share indices. For example  $(0, 1, 2) \rightarrow (0, 1, 2)$  denotes the identity as all shares are mapped to their original position. Note, however, that each of the six possible permutations has no impact on the claimed first-order security of this construction considering the  $(1, 0, 0)$ -robust probing model.

**Example 3** (Strong 8-bit Iterative S-Boxes). Example 1 and Example 2 indicated that iterative circuits are prone to transitional leakage. Therefore, using our transition-extended version of SILVER, we tried to examine all eight different S-box constructions. We first examined the combinational function of all such designs by  $(1, 0, 0)$ -robust probing model and verified with SILVER that considering only glitches, the combinational function is first-order secure and provides a uniform output sharing, i.e., its composition does not violate security properties. As stated, permuting the shares of the feedback signals should not affect the security if the probes are only extended due to glitches. This is also trivially confirmed by SILVER. However, such a permutation may have an effect when transitions are taken into account. Therefore, we evaluated all eight iterative designs for all six possible share permutations considering the  $(1, 1, 0)$ -robust probing model. The results of such evaluations are summarized in Table 1. Beside the result of first-order evaluations, we give the corresponding number of evaluated probe combinations and the execution time next to each evaluation result.

It can be seen that considering the identity  $(0, 1, 2) \rightarrow (0, 1, 2)$ , i.e., the original construction in [BGG<sup>+</sup>17], none of the constructed iterative S-boxes is secure when the probes are extended based on glitches and transitions. Interestingly, our extended version of SILVER reports no leakage for three designs  $SB_1$ ,  $SB_4$ , and  $SB_5$  when feedback shares are permuted as  $(0, 1, 2) \rightarrow (1, 2, 0)$ . The same holds for two designs  $SB_1$  and  $SB_4$  for permutation  $(0, 1, 2) \rightarrow (2, 0, 1)$ . Since these designs belong to different categories, we believe that it is a coincidence with no dependency on the underlying construction of these S-boxes. As a consequence, among the iterative designs introduced in [BGG<sup>+</sup>17] there is no certain design category combined with a specific permutation of feedback shares which guarantees the security in presence of glitches and transitions, independent of the employed

**Table 1:** First-order evaluation results of all 8-bit S-boxes reported by SILVER under the (1, 1, 0)-robust probing model, including the number of probe combinations and the execution time on a Windows 10 server with 96 cores and 256GB RAM.

S-Box	Type	Iter. #	Area GE	Share Permutation P: (0, 1, 2) →					
				(0, 1, 2)	(0, 2, 1)	(1, 0, 2)	(1, 2, 0)	(2, 0, 1)	(2, 1, 0)
SB <sub>1</sub>	Perm.	8	241	✗ <sup>3</sup> <sub>2.2 s</sub>	✗ <sup>19</sup> <sub>9.9 s</sub>	✗ <sup>3</sup> <sub>2.1 s</sub>	✓ <sup>108</sup> <sub>2.2 min</sub>	✓ <sup>116</sup> <sub>2.2 min</sub>	✗ <sup>11</sup> <sub>10.3 s</sub>
SB <sub>2</sub>	Matrix	2	446	✗ <sup>1</sup> <sub>1.5 min</sub>	✗ <sup>1</sup> <sub>1.5 min</sub>	✗ <sup>1</sup> <sub>1.5 min</sub>	✗ <sup>2</sup> <sub>10.7 min</sub>	✗ <sup>1</sup> <sub>1.3 min</sub>	✗ <sup>2</sup> <sub>10.2 min</sub>
SB <sub>3</sub>	Matrix	4	458	✗ <sup>1</sup> <sub>3.6 min</sub>	✗ <sup>1</sup> <sub>1.5 min</sub>	✗ <sup>1</sup> <sub>1.4 min</sub>	✗ <sup>1</sup> <sub>19.8 min</sub>	✗ <sup>1</sup> <sub>2.0 h</sub>	✗ <sup>1</sup> <sub>20.5 min</sub>
SB <sub>4</sub>	Feistel	5	363	✗ <sup>5</sup> <sub>1.2 s</sub>	✗ <sup>21</sup> <sub>22.8 min</sub>	✗ <sup>5</sup> <sub>1.5 s</sub>	✓ <sup>52</sup> <sub>1.2 h</sub>	✓ <sup>52</sup> <sub>1.5 h</sub>	✗ <sup>13</sup> <sub>21.8 min</sub>
SB <sub>5</sub>	Perm.	9	263	✗ <sup>3</sup> <sub>3.7 s</sub>	✗ <sup>3</sup> <sub>3.6 s</sub>	✗ <sup>3</sup> <sub>3.8 s</sub>	✓ <sup>56</sup> <sub>6.9 min</sub>	✗ <sup>3</sup> <sub>3.4 s</sub>	✗ <sup>11</sup> <sub>13.3 s</sub>
SB <sub>6</sub>	Matrix	4	420	✗ <sup>1</sup> <sub>37.6 s</sub>	✗ <sup>1</sup> <sub>34.5 s</sub>	✗ <sup>1</sup> <sub>37.0 s</sub>	✗ <sup>1</sup> <sub>7.4 min</sub>	✗ <sup>1</sup> <sub>4.6 min</sub>	✗ <sup>1</sup> <sub>7.7 min</sub>
SB <sub>7</sub>	Feistel	4	431	✗ <sup>7</sup> <sub>6.9 s</sub>	✗ <sup>7</sup> <sub>36.2 s</sub>	✗ <sup>7</sup> <sub>1.9 s</sub>	✗ <sup>7</sup> <sub>33.1 s</sub>	✗ <sup>7</sup> <sub>32.3 s</sub>	✗ <sup>7</sup> <sub>33.7 s</sub>
SB <sub>8</sub>	Feistel	8	332	✗ <sup>8</sup> <sub>2.7 s</sub>	✗ <sup>8</sup> <sub>1.1 min</sub>	✗ <sup>8</sup> <sub>2.7 s</sub>	✗ <sup>8</sup> <sub>1.1 min</sub>	✗ <sup>8</sup> <sub>2.3 min</sub>	✗ <sup>8</sup> <sub>1.1 min</sub>



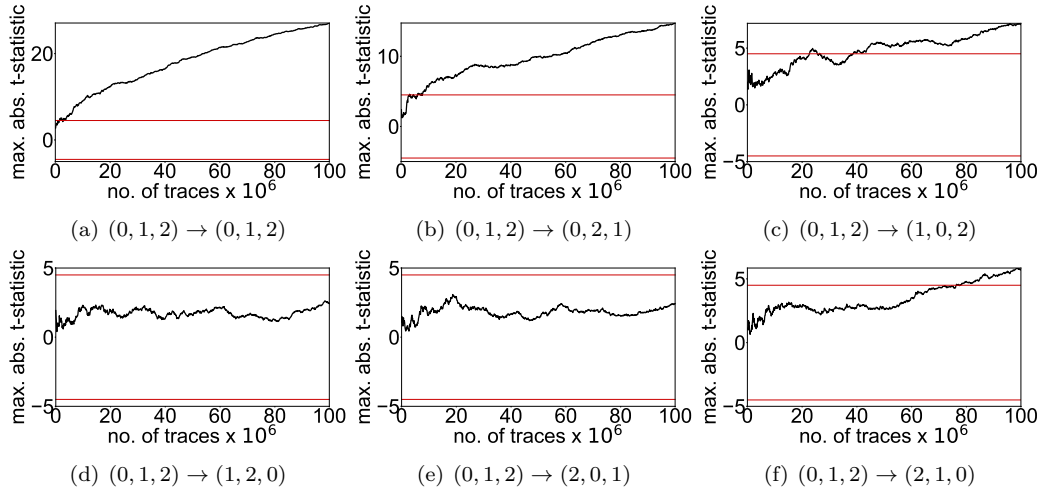
**Figure 6:** Iterative SB<sub>1</sub>, first-order fixed vs. random  $t$ -test results over time, using 100 million traces.

components (i.e., 4-bit S-boxes, linear layers, round function of the Feistel network).

#### 4.1 Experimental Analysis

As a sanity check, we examined the leakage of one of the iterative designs using real measurements. To this end, we have taken the iterative design of SB<sub>1</sub> (which is a first-order TI with 3 shares) and implemented the corresponding circuit on the Spartan-6 target Field-Programmable Gate Array (FPGA) of a SAKURA-G evaluation board [SAK] and recorded the dynamic power consumption by means of a digital oscilloscope at a sampling rate of 625 MS/s. During all measurements, the target architecture was being operated by a 6 MHz stable clock source. For the analysis, we applied the common Test Vector Leakage Assessment (TVLA) [GJJR11] approach on a set containing 100 million traces, measured while the circuit receives either a fixed or a random 3-share 8-bit input (to the 8-bit S-box). We have examined the same circuit for all six share permutations and limited our analyses to only first-order  $t$ -tests, as the underlying constructions are only supposed to provide first-order security.

The resulting  $t$ -statistics, which are shown in Figure 6 and Figure 7, precisely confirm our theoretical findings reported by our extended version of SILVER. More specifically, for



**Figure 7:** Iterative  $SB_1$ , first-order fixed vs. random  $t$ -test results over number of traces.

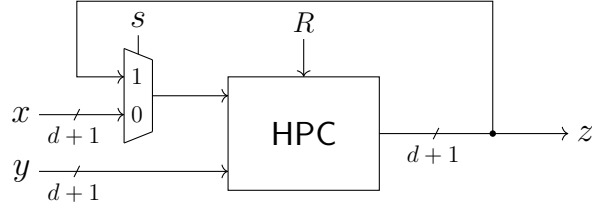
all designs that exhibit information leakage according to SILVER, we also observe leakage in the real measurements. Interestingly, for the original settings  $(0, 1, 2) \rightarrow (0, 1, 2)$  and  $(0, 1, 2) \rightarrow (0, 2, 1)$ , the  $t$ -statistics surpasses the 4.5 threshold by less than five million traces, while for other insecure cases, many more traces are required to observe a significant leakage. This experiment indicates and confirms that transitional leakage, which we have detected by our transition-extended version of SILVER, is not only a purely theoretical issue but can have a severe practical impact on the security of a design.

## 5 Case Study 2: HPC Gadgets

As given in Section 2.5, HPC2 gadgets offer trivial composability under the notion of PINI only in the  $(1, 0, 0)$ -robust probing model [CGLS21]. In order to be in conformity with the refined notion of O-PINI under transitions and glitches, a novel O-PINI2 multiplication gadget has been proposed in [CS21]. In contrast to the original HPC2 gadget, such an extended O-PINI2 gadget requires  $d$  additional fresh random bits and an extra register stage, with  $d$  referring to the security order of the gadget, i.e., operating on  $d + 1$  shares of each input.

In the course of this case study, we first verify the results from [CS21] with our extension of SILVER and confirm the reported issues related to transitional leakage when iterating PINI-secure gadgets as described. Moreover, we verify the  $(1, 1, 0)$ -robust probing security of an iterative design if the considered gadget is in conformity with the O-PINI notion. Second, we show that the proposed O-PINI security notion is only necessary for a scenario where the circuit's loop is made by a single register stage. Eventually, we show how to avoid transitional leakages even without the application of an O-PINI secure gadget but solely relying on the length of the circuit's loop.

**Example 4 (Original Design).** We first focus on the issues reported in [CS21]. More precisely, we consider the iterative circuit shown in Figure 8, where input shares  $x$  and  $y$  are given to an HPC 2-input multiplication gadget, which is iterated several times while using the gadget output shares  $z$  as new input shares instead of  $x$ . Following [CS21], we instantiated both, an HPC2 and an O-PINI2 gadget as the underlying 2-input multiplication gadget in Figure 8, and verified both constructions up to the third security order, i.e.,  $d \leq 3$ , using our extended version of SILVER. As a side note, there is an inherent latency *imbalance* in the design of any HPC2 and O-PINI2 gadget, i.e., the required refreshing of



**Figure 8:** Iterative circuit for Examples 4 and 5.

**Table 2:** Leakage verification results of iterated HPC2 and O-PINI2 gadgets under  $(1,1,0)$ -robust probing model, including the number of probe combinations and the execution time on a Windows 10 server with 96 cores and 256GB RAM.

Gadget	Area	Random.	Security	Order $d$	Complexity	
	[GE]	[bits]	[expected]	[achieved]	[# of probes]	[run time]
HPC2	104	1	1	0 ✗	8	0.01 s
HPC2	250	3	2	1 ✗	51	0.2 s
HPC2	459	6	3	2 ✗	1 862	53.8 min
O-PINI2	142	2	1	1 ✓	22	0.03 s
O-PINI2	309	5	2	2 ✓	1 035	2.7 min
O-PINI2	539	9	3	3 ✓	73 226	1.5 d

a single input introduces an additional cycle of latency for this input only. Hence, with respect to the generation of the output  $z$ , the input  $x$  has a lower latency compared to the other input  $y$ . Table 2 summarizes our evaluation results.

Our results indeed confirm the issues reported in [CS21], i.e., transitional leakage decreases the security order by one if the gadget is PINI secure but not O-PINI secure. Hence, in these cases,  $d$ -order security under the  $(1,1,0)$ -robust probing model is only satisfied if the gadget is at least  $(d+1)$ -order  $(1,0,0)$ -robust probing secure. When evaluating the HPC2 gadgets, we noticed that SILVER detects the leakage in the second clock cycle, when the feedback  $z$  is given as the new input instead of  $x$ . We show the details of this issue for  $d=1$  in Figure 9(a) which depicts half of the circuit, i.e., the part of the circuit which generates the first output share  $z^0$ .

For this example, SILVER identified a single probe  $\mathbf{P} = \{\mathbf{p}_0\}$  indicating first-order leakage. More precisely,  $\mathbf{p}_0$  is placed on an AND gate computing  $r \overline{x^0}$  during the second clock cycle when  $r$  is stored in a register. Note that we provided the tool with a sequence of primary inputs  $r$ , i.e., distinct  $r_{\{i\}} \in \mathbb{F}_2$  in cycle  $i$  indicating that the fresh mask is updated at every clock cycle.

Starting with the glitch-extension scheme,  $\mathbf{p}_0$  is extended as follows:

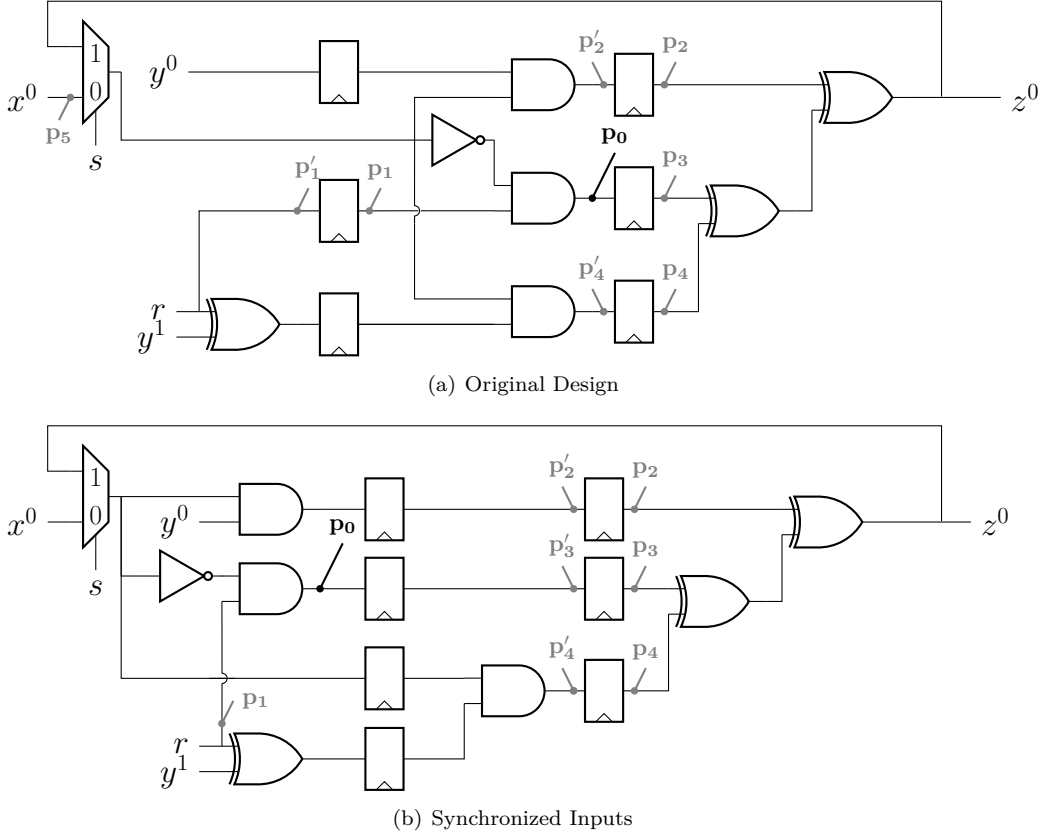
$$\mathbf{p}_0 \xrightarrow{(1,0,0)} \{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5\}, \quad \mathbf{P}_g = \{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5\},$$

where  $\mathbf{p}_1$  observes the register which stores  $r$  and  $\mathbf{p}_5$  the primary input  $x^0$ . During the next clock cycle,  $x^0$  is replaced by the feedback signal  $z^0$ , hence a transition between  $x^0$  and  $z^0$ . This leads to the following glitch-extended and transition-extended set of probes.

$$\mathbf{p}_1 \xrightarrow{(0,1,0)} \{\mathbf{p}_1, \mathbf{p}'_1\}, \quad \mathbf{p}_2 \xrightarrow{(0,1,0)} \{\mathbf{p}_2, \mathbf{p}'_2\}, \quad \mathbf{p}_3 \xrightarrow{(0,1,0)} \{\mathbf{p}_3, \mathbf{p}_0\}, \quad \mathbf{p}_4 \xrightarrow{(0,1,0)} \{\mathbf{p}_4, \mathbf{p}'_4\},$$

$$\mathbf{P}_{g,t} = \{\mathbf{p}_1, \mathbf{p}'_1, \mathbf{p}_2, \mathbf{p}'_2, \mathbf{p}_3, \mathbf{p}_0, \mathbf{p}_4, \mathbf{p}'_4, \mathbf{p}_5\}$$

Now, we consider the output after the first iteration, i.e.,  $z^0 = x^0 y^0 \oplus x^0 y^1 \oplus r_{\{0\}}$ , while the extended probes  $\{\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4\}$  observe  $(x^0 y^0, r_{\{0\}} x^0, x^0 (y^1 \oplus r_{\{0\}}))$ . Simultaneously,  $\mathbf{p}'_1$  observes the fresh mask at the second clock cycle, i.e.,  $r_{\{1\}}$ . Therefore, probe  $\mathbf{p}_0$



**Figure 9:** Single share computation of a first-order HPC2 2-input multiplication gadget.

leads to an observation including  $(r_{\{0\}}, r_{\{1\}}, x^0, x^0 y^0, r_{\{0\}} \bar{x}^0, x^0 (y^1 \oplus r_{\{0\}}))$ . This trivially leaks information about  $y$ . For an O-PINI2 gadget,  $z^0$  would be refreshed with another mask independent of  $r$  and stored in a register, which avoids such a first-order leakage by extending  $\mathbf{p}_0$  under the  $(1,1,0)$ -robust probing model.

**Example 5** (Delayed Feedback). As stated, the latency of  $x$  to  $z$  is one clock cycle. Therefore, the feedback is propagated at a point in time when the applied fresh mask is updated simultaneously. Hence, the transition-extended probes would capture the applied fresh mask and the feedback signal which is blinded by the same fresh mask, hence an undesired information leakage. A potential solution is to delay the feedback for (at least) one additional clock cycle in such a way that, when the transition between the old input  $x$  and the feedback signal  $z$  occurs, the fresh masks are updated (at least) one clock cycle before. We realize this by delaying input  $x$  one clock cycle to be synchronized with the other input  $y$ . This is shown in Figure 9(b).

Hence, a single probe  $\mathbf{p}_0$  would still be extended by glitches to  $\mathbf{P}_g = \{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5\}$ . At the third clock cycle,  $\mathbf{p}_5$  observes  $x_0$  while  $\{\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4\}$  observe the result of the first iteration, i.e.,  $(x^0 y^0, r_{\{0\}} \bar{x}^0, x^0 (y^1 \oplus r_{\{0\}}))$ . At the same clock cycle,  $\mathbf{p}_1$  observes two consecutive fresh masks  $r_{\{1\}}$  and  $r_{\{2\}}$ . This obviously does not lead to any leakage as the result of the first iteration is blinded by  $r_{\{0\}}$  which is not observed by  $\mathbf{p}_1$ . Placing any other single probe on this circuit does not lead to any first-order leakage under  $(1, 1, 0)$ -probing model. We have confirmed this by evaluating such a circuit (for  $d \leq 3$ ) with our extended version of SILVER.

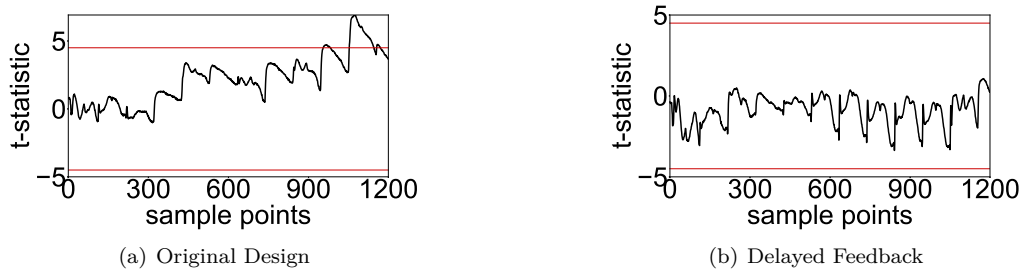
As a side note, such a synchronization of inputs of the HPC2 gadget would solve the

issue and at the same time keep its total latency of two clock cycles, in contrast to O-PINI2 multiplication gadget which has a latency of three clock cycles. As a result, if HPC gadgets are used in an iterative circuit as shown in Figure 8, transitional leakage does not degrade their security if the sequential loop (involving the feedback signal) consists of at least two register stages and essentially fresh masks are updated at every clock cycle.

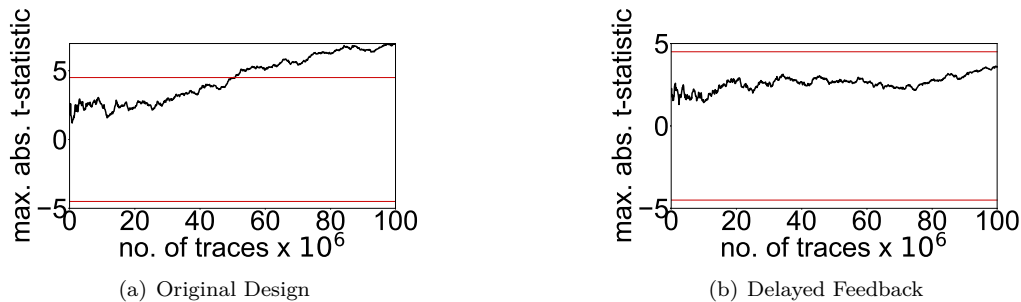
## 5.1 Experimental Analysis

Similar to the first case study (Section 4), we confirm our theoretical findings by means of an experimental analysis. We have used the same measurement setup, and implemented both designs of Example 4 and Example 5 for  $d = 1$ , collected 100 million traces recording the instantaneous power consumption during 5 iterations, and conducted the same evaluation, i.e., fixed-versus-random  $t$ -test.

The results, shown in Figure 10 and Figure 11, are inline with those reported by SILVER. Most importantly, we do not observe any first-order leakage from the design used in Example 5, while this is not the case for the design of Example 4. Note that the leakage detected for Example 4 is slightly above the threshold, i.e., more than 50 million measurements are required to observe the leakage. By this, we would like to highlight that with the theory and tools at hand (SILVER), we can detect the security flaws in such designs, but we cannot acquire any overview on their detectability and/or exploitability in practice.



**Figure 10:** Iterative HPC gadget, first-order fixed v. random  $t$ -test results over time, using 100 million traces.



**Figure 11:** Iterative HPC gadget, first-order fixed v. random  $t$ -test results over number of traces.

## 6 Conclusions

In this work, we present a novel methodology for modeling transition- and glitch-extended probes, enabling us to integrate the verification of transition-based leakage into SILVER, an existing software framework for formal verification of masked circuits which before was limited to perform verification in the  $(1, 0, 0)$ -robust  $d$ -probing model only, i.e., under the occurrence of glitches. With the integration of our methodology into SILVER, we enable designers to now also formally evaluate the security of hardware designs in the presence of glitches combined with transitions (i.e., in the  $(1, 1, 0)$ -robust  $d$ -probing model), which is highly relevant for constructing SCA-resilient iterative hardware designs.

For this, we present fundamental concepts to model and verify transition-based information leakage originating from memory recombinations, feedback loops, and transitions in primary inputs. We further demonstrate the relevance of our model extension by means of two different case studies. More precisely, the first case study demonstrates the power of the extended version of SILVER, for the first time enabling the verification and detection of security flaws in the iterative S-box designs introduced in [BGG<sup>+</sup>17].

Additionally, our second study confirms the composition flaws of HPC2 multiplication gadgets, as initially discussed in [CS21], when operated iteratively. In particular, this also allows us to show that some constructions proposed in [CS21] might be seen over-conservative with respect to security. Ultimately, for both case studies, we validate and confirm our findings (i.e., information leakage reported by our extended version of SILVER) by means of experimental leakage assessments.

While this work covers formal verification of SCA-resilience under glitches combined with transitions and is publicly available at [GitHub](#)<sup>6</sup>, we should stress that its ability to cover transitional leakage is limited to a certain form of circuits. More precisely, the transitions associated to the input sequences are covered as long as the probes are placed at the combinational circuit receiving such primary inputs. If the probes are placed on the combinational circuits which are not directly fed by the primary inputs, the transitional leakage originating from the input sequences might not be detected. Further, this tool does not yet cover any coupling-related leakage. As the detection of this would require additional routing information, the overall extension scheme cannot be performed on netlist level anymore. Nevertheless, a complete leakage verification should also take coupling effects into account. Hence, the automated verification under the  $(1, 1, 1)$ -robust  $d$ -probing model is a promising topic for future works.

## Acknowledgments

The work described in this paper has been supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and through the projects 393207943 GreenSec, 435264177 SAUBER, and 406956718 SuCCESS.

## References

- [BBC<sup>+</sup>19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In *ESORICS 2019*, volume 11735 of *Lecture Notes in Computer Science*, pages 300–318. Springer, 2019.

---

<sup>6</sup><https://github.com/Chair-for-Security-Engineering/SILVER/tree/transitional-leakage>



- [BBD<sup>+</sup>15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified Proofs of Higher-Order Masking. In *EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.
- [BBD<sup>+</sup>16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong Non-Interference and Type-Directed Higher-Order Masking. In *CCS 2016*, pages 116–129. ACM, 2016.
- [BDF<sup>+</sup>17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. In *EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–566, 2017.
- [BDM<sup>+</sup>20] Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. Tornado: Automatic Generation of Probing-Secure Masked Bitsliced Implementations. In *EUROCRYPT 2020*, volume 12107 of *Lecture Notes in Computer Science*, pages 311–341. Springer, 2020.
- [BGG<sup>+</sup>14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the Cost of Lazy Engineering for Masked Software Implementations. In *CARDIS 2014*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.
- [BGG<sup>+</sup>16] Erik Boss, Vincent Grosso, Tim Güneysu, Gregor Leander, Amir Moradi, and Tobias Schneider. Strong 8-bit Sboxes with Efficient Masking in Hardware. In *CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 171–193. Springer, 2016.
- [BGG<sup>+</sup>17] Erik Boss, Vincent Grosso, Tim Güneysu, Gregor Leander, Amir Moradi, and Tobias Schneider. Strong 8-bit sboxes with efficient masking in hardware extended version. *J. Cryptogr. Eng.*, 7(2):149–165, 2017.
- [BGI<sup>+</sup>18] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal Verification of Masked Hardware Implementations in the Presence of Glitches. In *EUROCRYPT 2018*, volume 10821 of *Lecture Notes in Computer Science*, pages 321–353. Springer, 2018.
- [BGN<sup>+</sup>14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-Order Threshold Implementations. In *ASIACRYPT 2014*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.
- [BGR18] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight Private Circuits: Achieving Probing Security with the Least Refreshing. In *ASIACRYPT 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 343–372. Springer, 2018.
- [Bil15] Begül Bilgin. *Threshold implementations : as countermeasure against higher-order differential power analysis*. PhD thesis, University of Twente, Enschede, Netherlands, 2015.
- [CBG<sup>+</sup>17] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does Coupling Affect the Security of Masked Implementations? In *COSADE 2017*, volume 10348 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2017.

- [CBR<sup>+</sup>15] Thomas De Cnudde, Begül Bilgin, Oscar Reparaz, Ventzislav Nikov, and Svetla Nikova. Higher-Order Threshold Implementation of the AES S-Box. In *CARDIS 2015*, volume 9514 of *Lecture Notes in Computer Science*, pages 259–272. Springer, 2015.
- [CGD18] Yann Le Corre, Johann Großschädl, and Daniel Dinu. Micro-architectural Power Simulator for Leakage Assessment of Cryptographic Software on ARM Cortex-M3 Processors. In *COSADE 2018*, volume 10815 of *Lecture Notes in Computer Science*, pages 82–98. Springer, 2018.
- [CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.
- [CGP<sup>+</sup>12] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of Security Proofs from One Leakage Model to Another: A New Issue. In *COSADE 2012*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.
- [CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
- [CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Transactions on Information Forensics and Security*, 15:2542–2555, 2020.
- [CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):136–158, 2021.
- [FGP<sup>+</sup>18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.
- [GHP<sup>+</sup>21] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-Design and Co-Verification of Masked Software Implementations on CPUs. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 1469–1468. USENIX Association, 2021.
- [GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for sidechannel resistance validation. In *NIST non-invasive attack testing workshop*, 2011.
- [GM18] Hannes Groß and Stefan Mangard. A unified masking approach. *J. Cryptogr. Eng.*, 8(2):109–124, 2018.
- [GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In *CT-RSA 2017*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017.
- [GMO01] Karine Gandolfi, Christophe Mourtél, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In *CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

- [HS13] Michael Hutter and Jörn-Marc Schmidt. The Temperature Side Channel and Heating Fault Attacks. In *CARDIS 2013*, volume 8419 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2013.
- [ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
- [KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
- [Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
- [KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - Statistical Independence and Leakage Verification. In *ASIACRYPT 2020*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.
- [MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):256–292, 2019.
- [MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In *CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.
- [MS06] Stefan Mangard and Kai Schramm. Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations. In *CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2006.
- [NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In *ICICS 2006*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.
- [NRS08] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. In *ICISC 2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2008.
- [RBN<sup>+</sup>15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In *CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.
- [SAK] SAKURA. Side-channel Attack User Reference Architecture. <http://satoh.cs.uec.ac.jp/SAKURA/index.html>.



# Chapter 5

## Novel and Securely Composable Hardware Modules for Masking

*In this chapter, we present all peer-reviewed publications accumulated during this thesis that relate to novel and securely composable hardware modules for masking. In this context, we present two papers published in the IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES) and one that was published in the proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS).*

### Contents of this Chapter

---

5.1	Generic Hardware Private Circuits – Towards Automated Generation of Composable Secure Gadgets . . . . .	133
5.2	Composable Gadgets with Reused Fresh Masks – First-Order Probing-Secure Hardware Circuits with only 6 Fresh Masks . . . . .	157
5.3	Low-Latency Hardware Private Circuits . . . . .	187

---

### 5.1 Generic Hardware Private Circuits – Towards Automated Generation of Composable Secure Gadgets

#### Publication Data

David Knichel, Pascal Sasdrich, and Amir Moradi. Generic Hardware Private Circuits – Towards Automated Generation of Composable Secure Gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):323–344, 2022

*This work is reproduced here with permission. This work is licensed under a Creative Commons Attribution 4.0 International License. Copyright is held by the authors. The author of this thesis is also an author of this research paper.*

**Content.** In this work, we present a methodology to transform any unprotected circuit into a PINI-composable hardware gadget in the first-order glitch-extended robust probing model. The transformation can simply be performed by means of the Boolean function description. We present two variants of our gadget construction. GHPC, in its standard variant, allows

to transform any vectorial Boolean function into a PINI-composable masked circuit which has a latency of two clock cycles and needs one random bit per coordinate function. With our low-latency variant,  $\text{GHPC}_{\text{LL}}$ , we can even lower the latency to a single clock cycle while the randomness requirements are increased to  $2^n$  per coordinate function where  $n$  is the number of unshared inputs. Next to presenting formal security arguments for all constructions, we provide extensive case studies and experimental leakage assessments.

**Contribution.** The author of this thesis is the principal author of this work. The author of this thesis likes to thank his co-authors for their contributions to this publication.

# Generic Hardware Private Circuits

## Towards Automated Generation of Composable Secure Gadgets

David Knichel<sup></sup>, Pascal Sasdrich<sup></sup> and Amir Moradi<sup></sup>

Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany

[firstname.lastname@rub.de](mailto:firstname.lastname@rub.de)

**Abstract.** With an increasing number of mobile devices and their high accessibility, protecting the implementation of cryptographic functions in the presence of physical adversaries has become more relevant than ever. Over the last decade, a lion's share of research in this area has been dedicated to developing countermeasures at an algorithmic level. Here, *masking* has proven to be a promising approach due to the possibility of formally proving the implementation's security solely based on its algorithmic description by elegantly modeling the circuit behavior. Theoretically verifying the security of masked circuits becomes more and more challenging with increasing circuit complexity. This motivated the introduction of security notions that enable masking of single gates while still guaranteeing the security when the masked gates are composed. Systematic approaches to generate these masked gates – commonly referred to as gadgets – were restricted to very simple gates like 2-input AND gates. Simply substituting such small gates by a secure gadget usually leads to a large overhead in terms of fresh randomness and additional latency (register stages) being introduced to the design.

In this work, we address these problems by presenting a generic framework to construct trivially composable and secure hardware gadgets for arbitrary vectorial Boolean functions, enabling the transformation of much larger sub-circuits into gadgets. In particular, we present a design methodology to generate first-order secure masked gadgets which is well-suited for integration into existing Electronic Design Automation (EDA) tools for automated hardware masking as only the Boolean function expression is required. Furthermore, we practically verify our findings by conducting several case studies and show that our methodology outperforms various other masking schemes in terms of introduced latency or fresh randomness – especially for large circuits.

**Keywords:** Masking, Generic and Composable Hardware Gadgets, Automated Masking, Side-Channel Analysis

## 1 Introduction

Even though Side-Channel Analysis (SCA) has been studied extensively by academic and industrial researchers, secure implementation of strong cryptographic implementations remains a challenging task. In the wake of the seminal description by Paul Kocher [Koc96], different approaches for countermeasures against SCA adversaries have been proposed. Among all candidates, *masking*, inspired by *secret sharing* concepts, is fascinating by its theoretical and sound security foundation [CJRR99] and has been applied manifold until today [ISW03, Tri03, NRS11, RBN<sup>+</sup>15, GMK17, GM18]. Unfortunately, not many of the proposed schemes have survived due to design flaws, inaccurate models, or invalid assumptions [MMSS19]. As a consequence, this trend of schemes whose assumptions have been proven invalid only confirms that, to the present day, design and implementation

of masking schemes is still a mostly manual, complex, and error-prone process, even for experienced security experts and hardware designers.

Facing such challenges, researchers recently started to focus on development of formal and accurate models of physical adversaries, hardware platforms, and execution environments as a mandatory foundation for formal verification and provably-secure schemes. In this light, formal verification of masked circuits is frequently conducted in the simple and abstract Ishai-Sahai-Wagner (ISW)  $d$ -probing security model [ISW03], given some basic assumptions on input and noise distribution, and its extension for more accuracy in the presence of physical defaults, e.g., glitches, transitions, and couplings [FGP<sup>+</sup>18].

Even though the introduction of a simple, yet practical, formal model accelerated verification, most security proofs are still limited to small circuits and masked gadgets only, mostly due to constraints in computational complexity. Naturally, modern approaches endeavor to extend formal verification to larger circuits through composition of formally verified gadgets, however, experience has shown that composition of secure gadgets is non-trivial and security proofs do not extend immediately.

Accordingly, several security notions for secure and trivial composition of masked gadgets have been proposed recently [BBD<sup>+</sup>15, BBD<sup>+</sup>16, CS20]. Although the security notions aim to assist in design and verification of larger circuits, creation of gadgets according to these rules in order to meet the requirements is still a challenge. More specifically, design of efficient gadgets under several optimization metrics, e.g., computational complexity, area demands, randomness requirements, performance in terms of latency and throughput, or higher-order protection still requires manual interaction and long-standing experience. To this end, the list of existing secure gadgets is limited, as most of them are hand-crafted, mainly focusing on protection of small gates, e.g., a 2-input AND [Tri03, BDF<sup>+</sup>17, FGP<sup>+</sup>18, CGLS20]. More importantly, these approaches usually are limited to atomic Boolean functions, e.g., AND and XOR, but do not provide a generic or automated approach to design secure, efficient, and trivially composable gadgets for different or arbitrary Boolean functions.

**Contributions.** In this work, we present a novel and generic framework that allows to easily construct trivially composable gadgets for arbitrary vectorial Boolean functions. In particular, relying on the glitch-extended probing adversary model and the secure composability notion of Probe-Isolating Non-Interference (PINI), our framework enables simple and generic construction of hardware private circuits and opens the possibility to transform any unprotected Boolean function into a first-order secure and composable gadget. In addition, backed by a thorough and sound theoretical security analysis and formal security arguments, our constructions enable efficient formal verification of entire cryptographic circuits and systems with respect to the PINI security notion. Eventually, we show practical relevance of our construction through experimental verification using different case studies and compare implementation results with respect to area, latency, and fresh randomness for various gadget constructions and related work.

**Outline.** Before we present our fundamental design principles based on Shannon’s Decomposition and provide a dedicated security analysis for our first-order secure construction schemes in Section 3, we first briefly present underlying assumptions and concepts in Section 2, including circuit representation, adversary model, security notions, and Boolean masking. In Section 4, we discuss and compare our proposed constructions to related works from literature, focusing on the metrics of area, latency, randomness, and composability. We further present different case studies to emphasize practical application of our concepts in Section 5 and experimentally confirm our theoretical security analyses based on leakage assessment for different PRESENT and AES designs. Eventually, we give a conclusion of the research conducted in this work in Section 6.



## 2 Background

### 2.1 Notation

Let us denote functions using sans-serif fonts, e.g.,  $F$ . Next, we denote random variables with uppercase letters, e.g.,  $X$ , while sets of random variables are given in bold, such as  $\mathbf{X}$ . Further, we use subscripts to indicate elements within a set while superscripts are used to denote (randomized) shares of random variables. Moreover, lowercase letters are used for the value of a random variable and bold lowercase letters indicate values for sets of variables accordingly. As a special case, the set of all shares of each random variable in  $\mathbf{X}$  is denoted as  $Sh(\mathbf{X})$  and  $P[\mathbf{X} = \mathbf{x}] = P[\mathbf{X}]$  denotes the joint probability that every  $X_i \in \mathbf{X}$  takes the value  $x_i \in \mathbf{x}$ . Moreover,  $\mathbf{X}^j$ ,  $j \geq 0$ , denotes the set containing all shares with index  $j$ . If  $\mathbf{S}$  is a set over arbitrary shares  $X_i^j$ , i.e.,  $\mathbf{S} \subset Sh(\mathbf{X})$ , then  $|\mathbf{S}|_i$  denotes the number of shares in  $\mathbf{S}$  that correspond to  $X_i$ .

### 2.2 Circuit Model

Throughout this work, any deterministic logic circuit  $C$  will be considered and modeled as a Directed Acyclic Graph (DAG)  $G_C = \{\mathcal{V}, \mathcal{E}\}$ , where  $\mathcal{V}$  gives the list of vertices and  $\mathcal{E}$  the list of edges of the DAG. Further, each vertex  $v \in \mathcal{V}$  represents a single combinational or sequential gate in the netlist while each edge  $e \in \mathcal{E}$  represents a single wire carrying an element from the finite field  $\mathbb{F}_2$ . In its entirety, a circuit realizes a vectorial Boolean function  $\mathbf{F} : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$  given its *coordinate functions*  $F_0, \dots, F_{m-1}$ , where  $\mathbf{F}$  is defined over its input  $\mathbf{X} \in \mathbb{F}_2^n$ .

**Encoded Circuit Model** As formalized by Ananth et al. [AIS18], a circuit compiler is a set of algorithms  $\{\text{Compile}, \text{Encode}, \text{Decode}\}$ , such that **Compile** is a deterministic algorithm that takes as input a circuit  $C$  and generates a randomized and encoded circuit  $\tilde{C}$ . Further, **Encode** is a probabilistic algorithm that takes as input a set of (secret) random variables  $\mathbf{X}$  and generates a shared representation  $Sh(\mathbf{X})$  with respect to some masking scheme (e.g., *Boolean masking*). Lastly, **Decode** is a deterministic algorithm that takes as input a shared representation  $Sh(\mathbf{Y})$  and reconstructs the according set of random variables  $\mathbf{Y}$ . Moreover, the circuit compiler has to satisfy *correctness* such that:

$$\text{Decode}(\tilde{C}(\text{Encode}(\mathbf{X}))) = C(\mathbf{X}), \forall \mathbf{X}.$$

### 2.3 Adversary Model

Before discussing common security notions and Boolean masking as theoretically sound countermeasure against SCA, we introduce the foundational  $d$ -probing adversary model which is used in modern literature to model side-channel adversaries and verify security of hardware circuits in presence of such adversaries.

**Traditional  $d$ -Probing Model.** In the traditional ISW  $d$ -probing model [ISW03], the adversarial strength is solely defined and limited by the number of probes that are granted to an adversary. Each probe can be used to observe and extract information carried on a single circuit<sup>1</sup> wire at a time. Assuming an ideal circuit, all gates and wires are updated simultaneously and each wire only carries the result of the driving gate under the current assignment of primary inputs. Then, depending on the number of granted probes, an adversary can combine information of up to  $d$  wires in the circuit to infer sensitive information. Further, a circuit is assumed to be secure under the  $d$ -probing adversary

<sup>1</sup>In the remainder of this work, we assume that the adversary is only able to observe an encoded circuit  $\tilde{C}$  while the **Encode** and **Decode** algorithms are unavailable for the adversary.

**Algorithm 1** glitch-extend**Input:** Probe  $P \in \mathbb{F}_2$ **Output:** Glitch-extended probe  $\mathbf{P}_{ext} \in \mathbb{F}_2^e, e > 0$ 


---

```

if  $P$  is placed on an output of a combinational gate then
     $\mathbf{P}_{ext} \leftarrow \bigcup_{0 \leq i < n} \text{glitch-extend}(P_i)$ 
    ▷ where  $P_i, 0 \leq i < n$  are
    all inputs to the combina-
    tional gate
else
    if  $P$  is placed on an output of a register or on a primary input then
         $\mathbf{P}_{ext} \leftarrow \{P\}$ 
    end if
end if

```

---

model if for any combination of up to  $d$  probes, the adversary is not able to learn about sensitive information (more details are given in Section 2.5).

**Probing in the Presence of Glitches.** Research has shown that physical circuits do not have an ideal behavior but physical defaults, e.g., glitches, transitions, and coupling [FGP<sup>+</sup>18], may cause unintentional leakages. In particular glitches, causing unintentional transitions on circuit wires due to non-ideal gates and different path delays, have been shown to introduce design vulnerabilities even for circuits that are secure under the standard  $d$ -probing adversary model [MPG05]. As a consequence, a robust  $d$ -probing model has been proposed [FGP<sup>+</sup>18] assuming a worst-case scenario under glitch-occurrence in physical circuits. More precisely, in contrast to the standard model, probes in the robust model are considered as *glitch-extended* and grant an adversary not only access to the signal on the probed wire but also any combination of stable driving signals (primary or registered inputs).

If  $P \in \mathbb{F}_2$  is a probe in the standard probing model, the corresponding glitch-extended probe can be derived by performing  $\text{glitch-extend}(P)$ , where  $\text{glitch-extend}(\cdot)$  can be defined recursively by returning either all extended probes on the input to the combinational gate, if  $P$  is placed on the output of such gate, or is defined by the identity function, if  $P$  is placed on an output of a register or a primary input. This is formally defined in Algorithm 1. A set of probes can be extended by union of the glitch extension of each probe.

## 2.4 Probe Simulatability

The concept of probe simulation helps to formally argue about dependencies between the probability distribution over probe observations and inputs to a masked (i.e., encoded) circuit.

**Definition 2.1** (Perfect Probe Simulation). *Given a set  $\mathbf{P} = \{P_0, P_1, \dots, P_{l-1}\}$  of  $l$  (glitch-extended) probes on a masked circuit  $\mathcal{C}$  with input  $Sh(\mathbf{X})$ ,  $\mathbf{P}$  can be perfectly simulated with a set over arbitrary shares  $\mathbf{S}$  iff there exists a simulator  $Sim$  such that for any shared input  $Sh(\mathbf{X})$  to  $\mathcal{C}$ , the probability distribution over  $\mathbf{P}$  and  $Sim(\mathbf{S})$  are equal, where  $Sim : \mathbb{F}_2^{|\mathbf{S}|} \mapsto \mathbb{F}_2^l$  with input  $\mathbf{S} \subseteq Sh(\mathbf{X})$  is a probabilistic polynomial time (p.p.t.) simulator.*

**Probe Propagation in Composed Circuits.** Experience has shown that composition of secure circuits, in the presence of  $d$  adversarial probes, may not result in secure composed circuits, given the same adversarial strength. More specifically, even though each circuit

separately can be proven to resist up to  $d$  adversarial probes, the effect of *probe propagation* may provide the adversary with more information than initially assumed. In particular, as downstream sub-circuits in a composed circuit usually process and combine results of upstream sub-circuits, placing up to  $d$  adversarial probes in those sub-circuits can provide information that, for isolated circuits, might only be obtainable by placing more than  $d$  probes, hence, virtually extending the adversarial strength beyond the limit of  $d$  probes. With the help of [Definition 2.1](#), we can formally define probe propagation as:

**Definition 2.2** (Probe Propagation). *A probe  $P \in \mathbb{F}_2^l$  is said to propagate into an input wire  $I \in \mathbb{F}_2$  iff  $I$  is required to perfectly simulate  $P$ , i.e.,  $I$  has to be in the simulation set  $S$  as defined in [Definition 2.1](#).*

## 2.5 Security Notions

Since the seminal introduction of the ISW  $d$ -probing adversary model [[ISW03](#)], many different security notions to analyze and verify the security of physical circuits have been proposed, in particular to ensure composition of secure circuits from provably secure sub-circuits. Below, we introduce the most common security notions, based on the consolidated definitions in [[DBR19](#)] and their generalization, unification, and extension as recently presented in [[KSM20](#)].

**Probing Security.** Granted access to internal values of a circuit through adversarial probes, an adversary may learn (partial) information on the processed secrets. Hence, in order to achieve probing security in the presence of up to  $d$  adversarial probes, any combination of up to  $d$  probes on internal values carried on wires must be statistically independent of the processed secrets. More specifically, this will limit the partial information any  $d$ -probing adversary can learn on the secrets, such that correct guessing and recovering of the sensitive information is impossible. More formally, probing security can be defined through [Definition 2.3](#).

**Definition 2.3** ( $d$ -Probing Security). *An encoded circuit  $\bar{C}$ , with secret input  $\text{Encode}(\mathbf{X})$ ,  $\mathbf{X} \in \mathbb{F}_2^n$ , is  $d$ -probing secure, if and only if for any observation  $\mathbf{Q}$  of  $t \leq d$  wires,  $\mathbf{X}$  is statistically independent of the observation, i.e.,  $P[\mathbf{Q}|\mathbf{X}] = P[\mathbf{Q}]$ .*

**Non-Interference.** While  $d$ -probing security purely focuses on the security of circuits in the presence of adversarial probes, the security notion of *Non-Interference (NI)* additionally targets the composition of masked circuits, usually considered as *gadgets*, such that security spans across the composed circuit instead of isolated gadgets only.

Through the concept of NI, flow of sensitive information is limited, although a  $d$ -probing adversary is still allowed to gain partial information on internal values and wires through adversarial probes. However, the original circuit, and in particular the original distribution of probed values, must not be distinguishable from a simulated distribution generated only based on the available partial information. As a consequence, each adversarial probe must be perfectly simulatable on partial information comprising a subset of all primary input shares limited by the security order  $d$ . More formally, the security notion of NI can be expressed through [Definition 2.4](#).

**Definition 2.4** ( $d$ -Non-Interference). *An encoded circuit  $\bar{C}$ , with secret input  $\text{Encode}(\mathbf{X})$ ,  $\mathbf{X} \in \mathbb{F}_2^n$ , is  $d$ -non-interfering if and only if for any observation  $\mathbf{Q}$  of  $t \leq d$  wires, there exists a set  $\mathbf{S}$  of input shares, with  $|\mathbf{S}|_i \leq t, \forall i$ , such that  $P[\mathbf{Q}|\mathbf{S}] = P[\mathbf{Q}|\text{Sh}(\mathbf{X})]$ .*

**Strong Non-Interference.** Unfortunately, the security notion of NI could not ensure *composability* of  $d$ -probing secure gadgets, due to the problem of *probe propagation* in

composed circuits. More precisely, composing gadgets may result in combination of partial information such that the placing of adversarial probes on downstream gadgets may propagate into upstream gadgets and grant the adversary access to partial information that otherwise could only be observed by placing more than  $d$  adversarial probes.

Hence, to correct deficiencies in the NI notion, once it comes to composition of secure gadgets, the stronger notion of Strong Non-Interference (SNI) was introduced. In particular, the concept of SNI intercepts probe propagation at the primary output of gadgets which again limits the partial information accessible through adversarial probes. As a consequence, each primary output of an SNI-secure gadget must be perfectly simulatable even without any partial information gained through adversarial probes. More formally, the security notion of SNI can be expressed through [Definition 2.5](#).

**Definition 2.5** (*d*-Strong Non-Interference). *An encoded circuit  $\bar{C}$ , with secret input  $\text{Encode}(\mathbf{X})$ ,  $\mathbf{X} \in \mathbb{F}_2^n$ , is *d*-strong-non-interfering if and only if for any observation  $\mathbf{Q}$  of  $t = t_1 + t_2 \leq d$  wires, with  $t_1$  being internal wires and  $t_2$  being output wires, there exists a set  $\mathbf{S}$  of input shares, with  $|\mathbf{S}|_i \leq t_1, \forall i$ , such that  $P[\mathbf{Q}|\mathbf{S}] = P[\mathbf{Q}|Sh(\mathbf{X})]$ .*

**Probe-Isolating Non-Interference.** Although the security notion of SNI resolves shortcomings in NI and allows secure composition of gadgets, this security notion, however, is rather conservative and inefficient in practice with respect to fresh entropy and circuit area. Moreover, Cassiers and Standaert [CS20] have shown that the concept of SNI is limited to single-output gadgets only, but does not scale for multi-output gadgets, again due to *probe propagation*. Although the concept of Multiple-Input-Multiple-Output SNI (MIMO-SNI) could fix the deficiencies, PINI was introduced as a more elegant and efficient solution.

In particular, the approach of PINI is inspired by trivial composition of linear functions (assuming Boolean masking) and the concept of *domain separation* as introduced in [GMK17]. More precisely, PINI-secure gadgets limit the propagation of adversarial probes with respect to share domains (also referred to as *circuit shares*), i.e., each share domain is separated and any adversarial probe will only propagate into its associated share domain. Given this, PINI-gadgets are trivially composable, similar to linear gadgets, regardless of the number of primary outputs. More formally, the security notion of PINI can be expressed through [Definition 2.6](#).

**Definition 2.6** (*d*-Probe-Isolating Non-Interference). *An encoded circuit  $\bar{C}$  with secret input  $\text{Encode}(\mathbf{X})$ ,  $\mathbf{X} \in \mathbb{F}_2^n$ , is *probe-isolating non-interfering* if and only if for any observation  $\mathbf{Q}$  of  $t = t_1 + t_2 \leq d$  wires, with  $t_1$  being internal wires and  $t_2$  being output wires, there exists a set of  $\mathbf{I}_{PI}$  primary input indices, with  $|\mathbf{I}_{PI}| \leq t_1$ , and  $\mathbf{I}_{PO}$  primary output indices, with  $|\mathbf{I}_{PO}| \leq t_2$ , such that  $\mathbf{Q}$  can be perfectly simulated by  $\mathbf{S} = Sh(\mathbf{X})^{\mathbf{I}_{PI} \cup \mathbf{I}_{PO}}$ .*

## 2.6 Boolean Masking

Due to its sound theoretical foundation, Boolean masking has been established as the most predominant approach to mitigate side-channel leakage in digital logic. In general,  $\text{Encode}$  for Boolean masking relies on concepts of secret sharing to split sensitive variables  $\mathbf{X}$  into Boolean shares  $\mathbf{X}^i$ , such that  $\mathbf{X} = \bigoplus_{i=0}^d \mathbf{X}^i$ , which allows simple masking of linear functions, but requires special considerations for non-linear operations.

Assuming that each Boolean share  $\mathbf{X}^i$  is independent of the secret  $\mathbf{X}$  and all other shares, a circuit implementing Boolean masking with  $d + 1$  shares can be evaluated securely even in the presence of  $d$  adversarial probes. However, as already mentioned, transient computations, i.e., glitches in hardware circuits, may recombine independent shares resulting in secret-dependent evaluations that may leak sensitive information. Hence, careful construction and layout of the masking scheme is imperative and a variety of different schemes has been proposed to ensure resistance even in the presence of glitches.

As a consequence, different hardware masking schemes have been proposed over the last years [ISW03, Tri03, NRS11, RBN<sup>+</sup>15, GMK17, GM18], most of them being extendable to higher-order protection and providing different trade-offs for computational and area complexity, memory requirement, latency, and randomness demand.

### 3 Generic Hardware Private Circuits (GHPC)

#### 3.1 Shannon Decomposition

In general, our construction for the design of generic and composable hardware private circuits for arbitrary Boolean functions utilizes the so-called *Shannon Decomposition* which was initially presented by Boole in [Boo48].

**Theorem 1** (Shannon Decomposition). *Any Boolean function  $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2$  can be written as*

$$F(X_0, X_1, \dots, X_i, \dots, X_{n-1}) = \overline{X_i} \cdot F(X_0, X_1, \dots, 0, \dots, X_{n-1}) \oplus X_i \cdot F(X_0, X_1, \dots, 1, \dots, X_{n-1}),$$

or in short:  $F = \overline{X_i} \cdot F|_{X_i=0} \oplus X_i \cdot F|_{X_i=1}$ , where  $F|_{X_i=0}$  and  $F|_{X_i=1}$  are called the *Shannon cofactors*.

Note that in the original definition, the Shannon cofactors were connected by a simple OR operation instead of an XOR. Correctness of both versions is nonetheless obvious, as by assigning a value to  $X_i$ , the corresponding Shannon cofactor is selected as output function, such that:

$$F = \begin{cases} F|_{X_i=0}, & \text{if } X_i = 0 \\ F|_{X_i=1}, & \text{if } X_i = 1 \end{cases} \quad (1)$$

Since  $F|_{X_i=0}$  and  $F|_{X_i=1}$  are again Boolean functions, this decomposition can be applied recursively, depending on arbitrary input variables. For example,  $F$  can be decomposed choosing  $X_i$  and  $X_j$ ,  $i \neq j$ , then leading to:

$$\begin{aligned} F &= \overline{X_i} \overline{X_j} \cdot F|_{X_i=0, X_j=0} \oplus \\ &\quad \overline{X_i} X_j \cdot F|_{X_i=0, X_j=1} \oplus \\ &\quad X_i \overline{X_j} \cdot F|_{X_i=1, X_j=0} \oplus \\ &\quad X_i X_j \cdot F|_{X_i=1, X_j=1}. \end{aligned} \quad (2)$$

In essence, translating a Shannon Decomposition of  $F$  into a logic circuit can be represented as a multiplexer (cascade) selecting the cofactors depending on the decomposition variables. For this, Equation 1 results in a 2-input multiplexer selecting depending on  $X_i$ , while Equation 2 results in a 4-input multiplexer selecting depending on  $X_i$  and  $X_j$ .

#### 3.2 Design

A high-level overview of our methodology for generating composable private circuits from unprotected circuits is depicted in Figure 1. Given an unprotected circuit  $C$  realizing a Boolean function  $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$  and knowing the function expression of  $F$ , our masking approach enables the construction of a first-order protected and composable hardware private circuit GHPC with two input and output shares under the PINI security notion (even in the presence of glitches). Further, the number of refreshing random bits of our approach is limited to only a single fresh random bit per coordinate function (i.e.,  $\mathbf{R} \in_R \mathbb{F}_2^m$ ). In fact, the result of the GHPC is a textbook sharing of each original coordinate

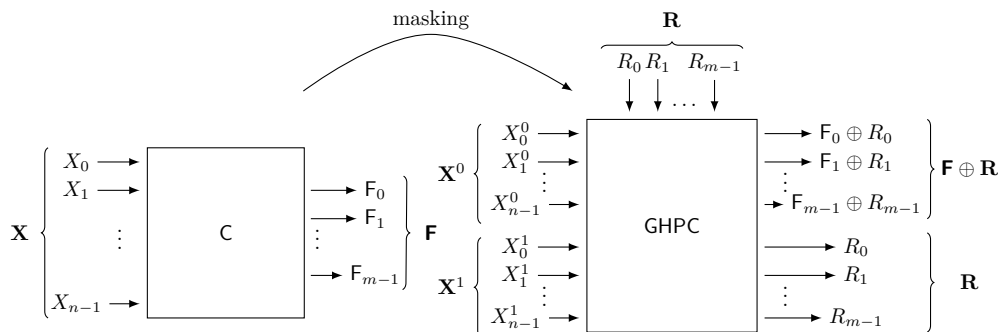


Figure 1: An overview of first-order masking

function  $F_i$ , i.e., each coordinate function is blinded by a different fresh mask  $R_i \in \mathbf{R}$ , while the second share is simply assigned the chosen random value  $R_i$  drawn from  $\mathbf{R}$ , hence, immediately ensuring uniformity and correctness of the sharing.

**Construction Principle.** Given the circuit  $C$ , our construction principle for translation into a GHPC allows to process and transform each coordinate function  $F_i$  with  $0 \leq i < m$  independently. For this, we will restrict the discussion of the construction principle to arbitrary single-output Boolean functions  $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ , as extension to vectorial Boolean functions is given trivially through application on each coordinate function separately.

In general, given a shared function expression obtained through *direct sharing* of the original function  $F$ , i.e.,  $F' = F(X_0^0 \oplus X_0^1, X_1^0 \oplus X_1^1, \dots, X_{n-1}^0 \oplus X_{n-1}^1)$ , our construction can be seen as Shannon Decomposition of  $F'$ , where each Shannon cofactor is blinded by  $R$  and  $F'$  is evaluated and decomposed based on shares from a single share domain. However, it is important to note that  $F'$  itself is never constructed explicitly, as a Shannon Decomposition based on one share allows to construct  $F'$  implicitly, as this, given simple Boolean masking, results in substituting any variable in the original function  $F$  with the corresponding (possibly negated) other share. For instance, it holds that if  $X_0^1 = 0$ ,  $X_1^1 = 1$ ,  $\dots$ , and  $X_{n-1}^1 = 1$ , then  $F'|_{\mathbf{x}^1=(0,1,\dots,1)} = F(X_0^0, \overline{X_1^0}, \dots, \overline{X_{n-1}^0})$ . Interestingly, in this case, the Shannon cofactors only depend on a single share domain, while selection of the correct computation, i.e., the selection of the correct cofactor, only depends on the other share domain.

Hence, the foundation of our construction is a multiplexer design that selects function evaluations restricted to one share, each evaluation blinded by the same random value  $R \in_R \mathbb{F}_2$ , as shown in Figure 2 and algorithmically described in Algorithm 2. Then, the selection of the correct evaluation only depends on the second share, e.g., shares from domain 1 for the given design. Note, however, that naming of share domains is not fixed but may be chosen arbitrarily, as long as the share domain naming is applied consistently throughout the entire design and the naming of the output domains is adopted accordingly (to ensure security under the PINI notion). Then, each blinded Shannon cofactor is stored in a register, and selected according to the other share subsequently. For this, the registers depicted in dashed lines (and denoted as  $\text{Reg}_{\text{pipe}}[]$  in Algorithm 2) ensure synchronization and enable a pipelined architecture, but do not have any effect on the security in the glitch-extended robust probing model in general. Eventually, as only the correct Shannon cofactor is enabled through an AND gate but all other factors are gated, summing up the values in the final register stage results in the correct but blinded output, hence, assigning  $R$  to the second share of the GHPC ensures correctness.

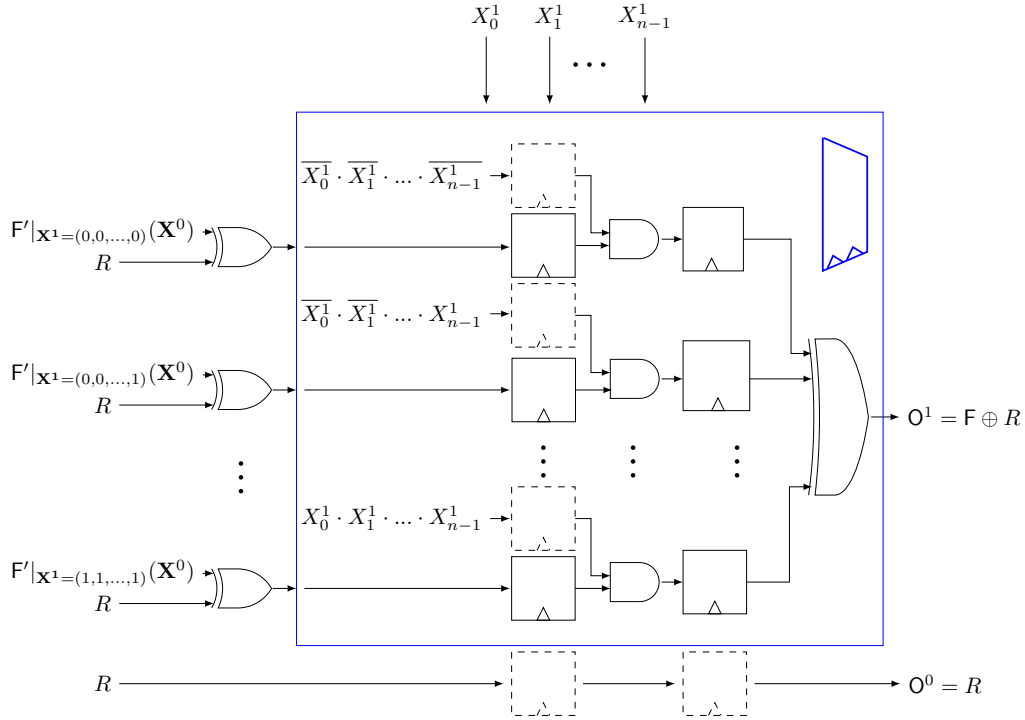


Figure 2: General GHPC design.

**Security Analysis.** In this section, we briefly prove the correctness and security of our construction under the notion of PINI, as stated in [Theorem 2](#), assuming the  $d$ -probing model with glitch-extended probes.

**Theorem 2.** *For an arbitrary circuit  $C$ , realizing a Boolean function  $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ , the transformation into a GHPC results in a correct and first-order PINI-secure circuit under the glitch-extended  $d$ -probing model.*

*Proof.*

*Correctness:* The correctness of the Shannon Decomposition directly implies that  $O^1 = F' \oplus R$ . As  $[\mathbf{X}^0, \mathbf{X}^1]$  is a valid sharing of  $\mathbf{X}$ , it follows that  $O^1 = F \oplus R$ , hence  $O = O^0 \oplus O^1 = F$ .

*PINI:* Considering [Figure 2](#) and [Algorithm 2](#), any extended probe on an input to the non-optional elements of the first register stage reveals all variables contributing to  $T_i \leftarrow \text{Reg}[F' |_{\mathbf{X}^1 = \text{BIN}(i)} \oplus R]$  for a fixed  $0 \leq i < 2^n - 1$ , i.e., the joint distribution over  $[R, \mathbf{X}^0]$ , which can be perfectly simulated with shares restricted to share domain 0 and drawing  $R \in_R \mathbb{F}_2$ . Further, any extended probe on the input of the second register stage reveals every stable variable contributing to  $M_i \leftarrow \text{Reg}[S_i \cdot T_i]$ , which translates to a leakage of the joint distribution over  $[T_i, \mathbf{X}^1]$ . Due to the blinding with  $R$ ,  $T_i$  can be simulated by drawing  $T_i \in_R \mathbb{F}_2$ . Hence, every observation can be simulated with shares restricted to domain 1. Eventually, placing an extended probe on the output  $O^1$  reveals the joint distribution over  $[M_0, M_1, \dots, M_{2^n-1}]$ . Here, depending on which input  $T_i$  is selected as the output, each observation will be of the form  $[0, \dots, 0, M_i, 0, \dots, 0]$ , i.e., a vector where all coordinates are zero except the one that shows a function perfectly blinded by  $R$ . This is due to the fact that by construction of the multiplexer design, only one second-stage

**Algorithm 2** GHPC**Input:** input shares  $\mathbf{X}^0, \mathbf{X}^1 \in \mathbb{F}_2^n$ , fresh randomness  $R \in \mathbb{F}_2$ **Output:**  $\mathbf{F} \in \mathbb{F}_2^2$ ,  $\mathbf{F} = [F^0, F^1] = [R, F \oplus R]$ 


---

```

 $O^1 \in \mathbb{F}_2, O^1 \leftarrow 0$ 
 $O^0 \leftarrow \text{Reg}_{\text{pipe}}[\text{Reg}_{\text{pipe}}[R]]$  ▷ Computation of  $O^0$ 
for  $\forall i \in \{0, \dots, 2^n - 1\}$  do ▷ Computation of  $O^1$ 
     $S_i \leftarrow \text{Reg}_{\text{pipe}}[\text{PRODUCT}(i, \mathbf{X}^1)]$ 
     $T_i \leftarrow \text{Reg}[F^1 |_{\mathbf{X}^1 = \text{BIN}(i)} \oplus R]$  ▷  $\text{BIN}(i)$  is the binary representation of  $i$ 
     $M_i \leftarrow \text{Reg}[S_i \cdot T_i]$ 
     $O^1 \leftarrow O^1 \oplus M_i$ 
end for

function  $\text{PRODUCT}(V \in \mathbb{F}_2^n, \mathbf{X} \in \mathbb{F}_2^n)$ 
     $P \in \mathbb{F}_2, P \leftarrow 1$ 
    for  $\forall i \in \{0, \dots, n - 1\}$  do
        if  $[\text{BIN}(V)]_i = 1$  then ▷  $[\text{BIN}(V)]_i$  is the  $i$ -th bit of the binary representation of  $V$ 
             $P \leftarrow P \cdot X_i$ 
        else
             $P \leftarrow P \cdot \overline{X_i}$ 
        end if
    end for
    return  $P$ 
end function

```

---

register contains the selected input — all others contain zero. The resulting vector can be perfectly simulated by drawing a fresh random bit and placing it at the right position  $i$  of the vector. However, note that in this example, the position  $i$  depends on the shares from share domain 1, hence, in order to provide PINI-security, exchanging the indices of the circuit output shares is not allowed. Eventually, every extended probe on the output  $O^0$  will only reveal a fresh random bit  $R$ , i.e., no information about the original input and/or output.  $\square$

**Examples.** In Figure 3, the designs resulting from masking a 2-input AND (Figure 3a) and a 3-input AND (Figure 3b) are given as examples. For this, we would like to highlight the clocked multiplexer symbol used in these figures, which refer to the same module identified by a blue border in Figure 2. As previously explained, the inputs to the multiplexer can be simply derived by inserting every combination of negated/non-negated shares from domain 0 into  $F = AB$ . This results in  $2^{n=2}$  input functions for the multiplexer design realizing a 2-input AND and in  $2^{n=3} = 8$  input functions for the 3-input AND. As an extra verification, we checked these designs with SILVER [KSM20] – a software tool for formal verification of masked circuits – which confirmed our theory by reporting first-order security under the PINI notion in the robust probing model.

### 3.3 Reducing the Latency

If desired, in order to reduce the overall latency, the number of register stages in the design can be reduced to a single stage.



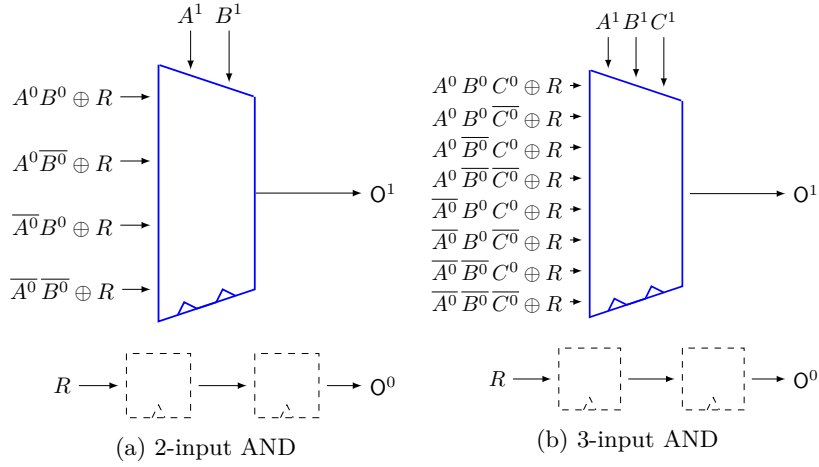


Figure 3: Examples for first-order PINI-secure GHPC constructions.

**Algorithm 3** GHPC<sub>LL</sub>**Input:** input shares  $\mathbf{X}^0, \mathbf{X}^1 \in \mathbb{F}_2^n$ , fresh randomness  $\mathbf{R} \in \mathbb{F}_2^{2^n}$ **Output:**  $\mathbf{F} \in \mathbb{F}_2^{2^n}$ ,  $\mathbf{F} = [\mathbf{F}^0, \mathbf{F}^1] = [R, \mathbf{F} \oplus R]$ , with  $R \in \mathbf{R}$ 


---

```

 $R \in \mathbb{F}_2, R \leftarrow 0$ 
 $O^1 \in \mathbb{F}_2, O^1 \leftarrow 0$ 
for  $\forall i \in \{0, \dots, 2^n - 1\}$  do
     $S_i \leftarrow \text{Reg}_{\text{pipe}}[\text{PRODUCT}(i, \mathbf{X}^1)]$   $\triangleright$  PRODUCT(.) as defined in Algorithm 2
     $T_i \leftarrow \text{Reg}[\mathbf{F}^1 |_{\mathbf{X}^1 = \text{BIN}(i)} \oplus R_i]$ 
     $M_i \leftarrow S_i \cdot T_i$ 
     $R \leftarrow R \oplus S_i \cdot R_i$ 
     $O^1 \leftarrow O^1 \oplus M_i$ 
end for
 $O^0 \leftarrow \text{Reg}[R]$ 

```

---

**Construction Principle.** For this, every input to the multiplexer must be blinded by a different freshly drawn random mask, as only this way, the second register stage in the first circuit share can be entirely omitted. For the output of circuit share 0, i.e., for  $O^0$ , the corresponding randomness has to be selected by a standard multiplexer before it is stored in a register. The resulting design with one register stage – referred to as GHPC<sub>LL</sub> in the following – can be seen in Figure 4 and in Algorithm 3. The architecture of circuit share 1 uses the same multiplexer architecture as presented in Figure 2, but with the second register stage omitted. Hence, this design has a reduced latency of 1 clock cycle, while expanding the demand for fresh randomness to  $2^n$  bits.

**Security Analysis.** Again, we briefly prove the correctness and security under the notion of PINI, as stated in Theorem 3, assuming the glitch-extended  $d$ -probing model.

**Theorem 3.** For an arbitrary circuit  $C$ , realizing a Boolean function  $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ , the transformation into a GHPC<sub>LL</sub> results in a correct and first-order PINI-secure circuit under the glitch-extended probing model.

*Proof.*

*Correctness:* Following the same argumentation as for Theorem 2,  $O^1$  outputs  $F \oplus R_i$  with  $i \in \{0, \dots, 2^n - 1\}$  and for any valid input sharing. As by construction,  $O^0$  equals  $R_i$ ,

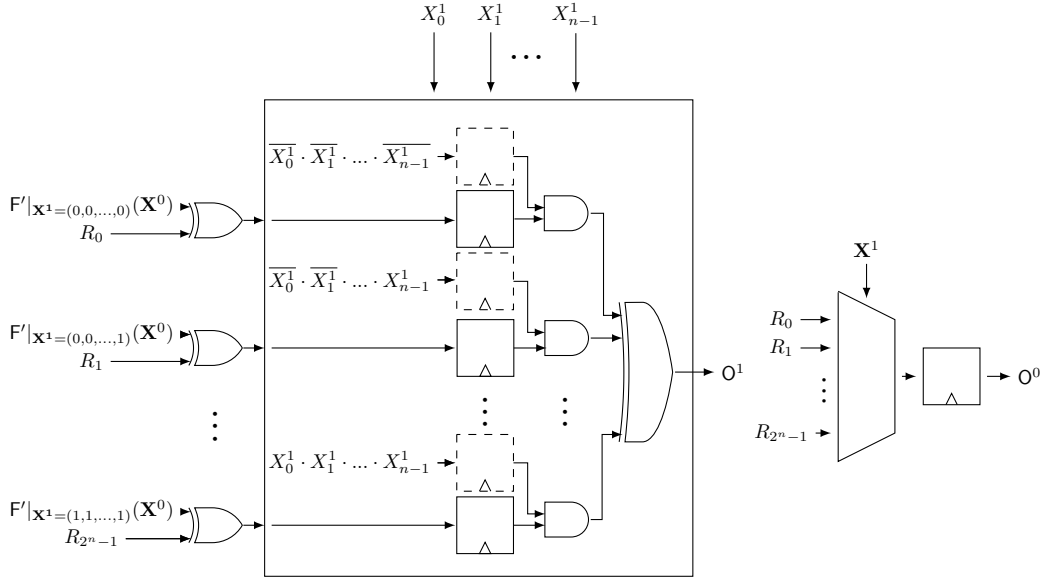


Figure 4: General GHPC<sub>LL</sub> design with reduced latency.

correctness of this masking is fulfilled.

*PINI*: Considering Figure 4 and Algorithm 3, the joint distribution  $[\mathbf{X}^0, R_i]$  using a probe on the input of  $T_i \leftarrow \text{Reg}[F' |_{\mathbf{x}^1 = \text{BIN}(i)} \oplus R_i]$  can be simulated using only shares from domain 0 and drawing  $R_i \in_R \mathbb{F}_2$ . An extended output probe on  $O^1$  will observe a joint distribution of the form  $[\mathbf{X}^1, T_0, T_1, \dots, T_{2^n-1}]$ , which can be simulated with  $\mathbf{X}^1$  and drawing  $2^n$  fresh random bits as  $T_i \in_R \mathbb{F}_2, \forall 0 \leq i < 2^n - 1$ . Due to the output register, a probe on  $O^0$  can be perfectly simulated using one fresh random bit, while each internal probe (on  $R$ ) is perfectly simulatable by fresh randomness and shares drawn only from share domain 1, as the whole circuit share does not involve computation on shares from share domain 0.  $\square$

## 4 Comparisons

In this section, we briefly discuss and compare our proposed constructions to state-of-the-art masking schemes with respect to common metrics, such as latency, demand for fresh randomness, area consumption, and composability of gadgets. To this end, Table 1 lists recent approaches from literature and their application to hardware circuits. In particular, we align our discussion and comparison by focusing on basic non-linear gates, i.e., 2-input AND gates, which are commonly used to create secure and composable gadgets required for construction of larger circuits. Further, we extend our discussion by comparing different techniques with respect to larger circuits, particularly using the PRESENT, PRINCE, Skinny, Prøst, Rectangle, Class-13, and AES S-boxes as illustrating examples. For the 4-bit S-boxes, we realized the corresponding descriptions given in [CGLS20] which are optimized with respect to the number of cascaded 2-input AND gates, i.e., favoring HPC1 and HPC2 as instantiated gadget. For the AES S-box, we considered the design given in [BP12], where – based on a tower field representation – a low-depth circuit has been constructed. It contains two isomorphisms at the start and end of the  $\text{GF}(2^8)$  inversion, and excluding the XOR gates, has at most 4 cascaded 2-input AND gates, which also is in favor of HPC1 and HPC2. In addition, we also include results reported in [GIB18], proposing a generic approach for low-latency masking which has been applied to the AES S-box considering different low-latency constructions.

Table 1: Comparison of different first-order masking schemes.  
(using Synopsis Design Compiler and UMC 180 standard cell library)

Target	Scheme	Func.		Latency	Rand.	Area	Compos.	Ref.
		$n$	$m$	[cycle]	[bit]	[GE]	notion	
AND2	DOM			2	1	56	SNI	[FGP <sup>+</sup> 18]
	HPC1			2	2	94	PINI	[CGLS20]
	HPC2	2	1	2	1	66	PINI	[CGLS20]
	GHPC			2	1	82	PINI	<b>new</b>
	GHPC <sub>LL</sub>			1	4	59	PINI	<b>new</b>
PRESENT S-box	HPC1			3	8	403	PINI	[CGLS20]
	HPC2			3	4	320	PINI	[CGLS20]
	GHPC <sub>LL</sub> -AND	4	4	2	16	310	PINI	<b>new</b>
	GHPC			2	4	1308	PINI	<b>new</b>
	GHPC <sub>LL</sub>			1	64	959	PINI	<b>new</b>
PRINCE S-box	HPC1			4	12	645	PINI	[CGLS20]
	HPC2			4	6	467	PINI	[CGLS20]
	GHPC <sub>LL</sub> -AND	4	4	2	24	445	PINI	<b>new</b>
	GHPC			2	4	1384	PINI	<b>new</b>
	GHPC <sub>LL</sub>			1	64	987	PINI	<b>new</b>
Skinny S-box	HPC1			4	8	467	PINI	[CGLS20]
	HPC2			4	4	301	PINI	[CGLS20]
	GHPC <sub>LL</sub> -AND	4	4	2	16	288	PINI	<b>new</b>
	GHPC			2	4	1232	PINI	<b>new</b>
	GHPC <sub>LL</sub>			1	64	951	PINI	<b>new</b>
Prøst S-box	HPC1			3	8	432	PINI	[CGLS20]
	HPC2			3	4	309	PINI	[CGLS20]
	GHPC <sub>LL</sub> -AND	4	4	2	16	302	PINI	<b>new</b>
	GHPC			2	4	1225	PINI	<b>new</b>
	GHPC <sub>LL</sub>			1	64	952	PINI	<b>new</b>
Rectangle S-box	HPC1			3	8	439	PINI	[CGLS20]
	HPC2			3	4	319	PINI	[CGLS20]
	GHPC <sub>LL</sub> -AND	4	4	2	16	311	PINI	<b>new</b>
	GHPC			2	4	1229	PINI	<b>new</b>
	GHPC <sub>LL</sub>			1	64	962	PINI	<b>new</b>
Class-13 S-box	HPC1			3	8	432	PINI	[CGLS20]
	HPC2			3	4	304	PINI	[CGLS20]
	GHPC <sub>LL</sub> -AND	4	4	2	16	303	PINI	<b>new</b>
	GHPC			2	4	951	PINI	<b>new</b>
	GHPC <sub>LL</sub>			1	64	933	PINI	<b>new</b>
AES S-box	CMS			5	54	2530	-	[CRB <sup>+</sup> 16]
	DOM			8	18	2851	-	[GMK17]
	GLLM			1	2048	60730 <sup>2</sup>	-	[GIB18]
	GLLM			2	416	6740 <sup>2</sup>	-	[GIB18]
	HPC1	8	8	5	68	3504	PINI	[BP12]
	HPC2			5	34	2452	PINI	[BP12]
	GHPC <sub>LL</sub> -AND			4	136	2376	PINI	<b>new</b>
	GHPC			2	8	77145	PINI	<b>new</b>
	GHPC <sub>LL</sub>			1	2048	64111	PINI	<b>new</b>

<sup>2</sup>These designs were synthesized using a different UMC 90 nm process technology.

Further, note that the descriptions given in [CGLS20] are without considering pipeline registers to synchronize the inputs of each gate. Therefore, in order to provide a fair comparison, all performance figures reported in Table 1 are for non-pipeline designs. Besides, similar to the state of the art, we did not include the area required for generation of fresh masks in the reported area footprints. All area results have been obtained by

synthesizing the Hardware Description Language (HDL) code of the design using Synopsis Design Compiler and UMC 180 nm standard cell library, unless indicated otherwise.

**Latency.** Since the final latency of our constructions does not depend on the underlying Boolean function and its algebraic degree, its application on larger functions leads to a higher efficiency with respect to latency compared to other approaches. Certainly, for small circuits and simple Boolean functions, e.g., a 2-input AND gate, hand-crafted and optimized gadgets might be more efficient in terms of latency. However, as our approach easily scales for larger functions, even with high algebraic degree, e.g., for an entire AES S-box, construction of masked circuits with low latency becomes feasible. More specifically, focusing on our low-latency approach  $\text{GHPC}_{\text{LL}}$  (at cost of additional randomness), our secure constructions outperform all other schemes listed in Table 1 in terms of latency. More precisely, to the best of our knowledge, the  $\text{GHPC}_{\text{LL}}$  is the only first-order composable construction with one clock cycle latency.

Although Groß et al. proposed a generic low-latency masking approach in [GIB18], the resulting AES S-box constructions certainly have a comparable latency (along with demand for fresh randomness and area), but do not result in a composable design but only focus on proving a probing-secure construction.

**Randomness.** In contrast to latency, the demand for fresh randomness of our constructions is mainly governed by the underlying Boolean function. The number of required fresh random bits  $r$  per circuit evaluation is given as

$$r_{\text{GHPC}} = m, \quad r_{\text{GHPC}_{\text{LL}}} = m \cdot 2^n.$$

More precisely, for GHPC this number is independent of the number of inputs  $n$  but only depends on the number of outputs  $m$  of the underlying Boolean function  $\mathbf{F} : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ . Hence, for large functions, such as 4-bit or 8-bit S-boxes, this results in randomness-efficient designs. This view for sure changes when lower latency is favorable, i.e.,  $\text{GHPC}_{\text{LL}}$  whose required fresh randomness depends on both  $m$  and  $n$ .

Then again, efficiency of our approach, in terms of required fresh randomness, does not change with the optimizations done on the implementation of the Boolean function. Taking the HPC1 and HPC2 of the S-boxes covered by Table 1, the foundational S-box implementations have been optimized through application of SAT solvers in order to reduce the latency and number of 2-input AND gates [CGLS20, BP12]. However, for our approach, the number of random bits is independent of how the Boolean function is realized. Instead, it only depends on its number of input and output bits. For this, our approach is particularly suitable for integration into EDA tools and automated integration of masking countermeasures into logic circuits.

**Area.** Besides latency and demand for fresh randomness during execution, the footprint in terms of area of the resulting design is an often considered metric in evaluation of efficiency and expense of a final design. In this regard, reduction in area usually can be traded for increasing latency and demand for fresh randomness.

In turn, this implies that our proposals optimized for low-latency ( $\text{GHPC}_{\text{LL}}$ ) and low-randomness (GHPC) naturally are outperformed by hand-crafted and sophisticatedly optimized gadgets and constructions. However, observing that our  $\text{GHPC}_{\text{LL}}$  construction for a 2-input AND not only provides best results in latency but also is smaller than all related (PINI-secure) hand-crafted gadgets, we opted to instantiate all S-box constructions provided in Table 1 with our  $\text{GHPC}_{\text{LL}}$  gadget instead of HPC1 or HPC2. Given this, we can observe that all our  $\text{GHPC}_{\text{LL}}$ -AND S-box constructions outperform related designs in terms of area (and even latency to some extent), but at cost of additional randomness that is required for secure execution. Construction of larger  $\text{GHPC}_{\text{LL}}$ -AND gadgets (3- or

4-bit input) to be used in the implementation of the S-boxes is also possible, but since the S-box descriptions we have in hand are not optimized to efficiently use such large-input gates, we have not included such cases in the presented results.

**Composability.** Eventually, our construction allows to build secure and composable hardware gadgets from arbitrary Boolean functions. As a consequence, even entire S-boxes, as shown in Table 1 can be transformed into securely-composable gadgets under the PINI notion. However, in contrast to the existing designs focusing on construction of secure circuits through composition of secure AND and XOR gates, our approach efficiently scales for arbitrary Boolean functions. For instance, AES S-box designs presented in [GMK17, CRB<sup>+</sup>16] rely on a careful instantiation of secure 2-input (and larger) AND gates. However, as these gadgets are not trivially composable, the resulting S-box circuit is indeed probing secure, but does not necessarily provide composability.

In contrast to this, our approach always results in PINI-secure gadgets, independent of the underlying function, allowing to construct gadgets even for larger circuits such as the AES S-box. As a result, we can conclude that due to its flexibility, our approach provides a clear road map for automatization of masking arbitrary circuits through generation of composable secure gadgets. It is true that a secure variant of any circuit can be constructed by HPC1 and HPC2 2-input AND (and XOR) gadgets, but there is a lower bound for the latency of the resulting circuit – which is of crucial importance in hardware designs – defined by the algebraic degree of the components of the underlying cryptographic function. However, our scheme uncouples this dependency while maintaining the same generality.

## 5 Case Studies

Below, we present the experimental results obtained when applying our construction principle to different block cipher implementations.

### 5.1 Target Device and Measurement Setup

The analyses have been conducted on a SAKURA-G board [SAK], where a Spartan-6 Field-Programmable Gate Array (FPGA) is embedded to host cryptographic cores. For all case studies given in the remainder of this section, the power consumption traces of the target FPGA have been collected by monitoring the voltage drop over a  $1\ \Omega$  resistor placed in the Vdd path amplified by an on-board AC amplifier. During the measurements performed by a digital oscilloscope at the sampling rate of 500 MS/s, the implemented cryptographic core was supplied by a stable and jitter-free clock source at the frequency of 6 MHz.

**PRNG.** For the generation of each fresh random bit, we constructed a 31-bit Linear-Feedback Shift Register (LFSR) with the feedback polynomial  $x^{31} + x^{28} + 1$ , which has a maximum cycle of  $2^{31} - 1$  with only two taps [WM12]. Each LFSR is initialized by an arbitrary value right after the FPGA power-up, making sure that no LFSR is entirely filled by zero, and there is no common initialization value for two LFSRs.

### 5.2 Byte-Serial AES

For our first case study, we opted to implement a first-order secure AES encryption based on the byte-serial architecture of Moradi et al. [MPL<sup>+</sup>11] with only minor modifications in the control logic due to the increased S-box latency. It is worth to highlight that all our HDL designs of the case studies are provided in the GitHub: <https://github.com/Chair-for-Security-Engineering/GHPC>.

**Design.** For this, a single GHPC AES S-box is instantiated and shared between data path and key expansion circuits, requiring both, data and key to be shared using  $d+1 = 2$  shares. Further, due to the two-cycle latency of the GHPC S-box design, and this component being the bottleneck of the architecture, we opted to include all pipelining registers to enable processing of all byte substitutions within 22 cycles, i.e., 2 cycles initial latency, 16 cycles for the round function S-box computations, and 4 cycles for the key expansion. Further, shifting of rows and mixing of columns is done in one respectively four cycles, while the key is updated simultaneously. In total, a single first-order secure AES round function (including key expansion) requires 23 cycles, resulting in a total of 230 cycles for an AES-128 encryption. Note that, although mixing of columns is omitted in the last round, expansion of the final post-whitening key is stalling the final round computation. We also provide a generic HDL description of our architecture in the GitHub: <https://github.com/Chair-for-Security-Engineering/GHPC> which allows to select GHPC or GHPC<sub>LL</sub> as the underlying design while adjusting the S-box, control logic, and randomness automatically (see more details in Appendix A). However, we considered only the GHPC design in our experimental analyses due to the similarity of the results.

Then, as given in Table 2, our entire first-order AES encryption architecture has a size of 86.3 kGE, using an 180 nm cell library while it requires only 8-bit fresh randomness per

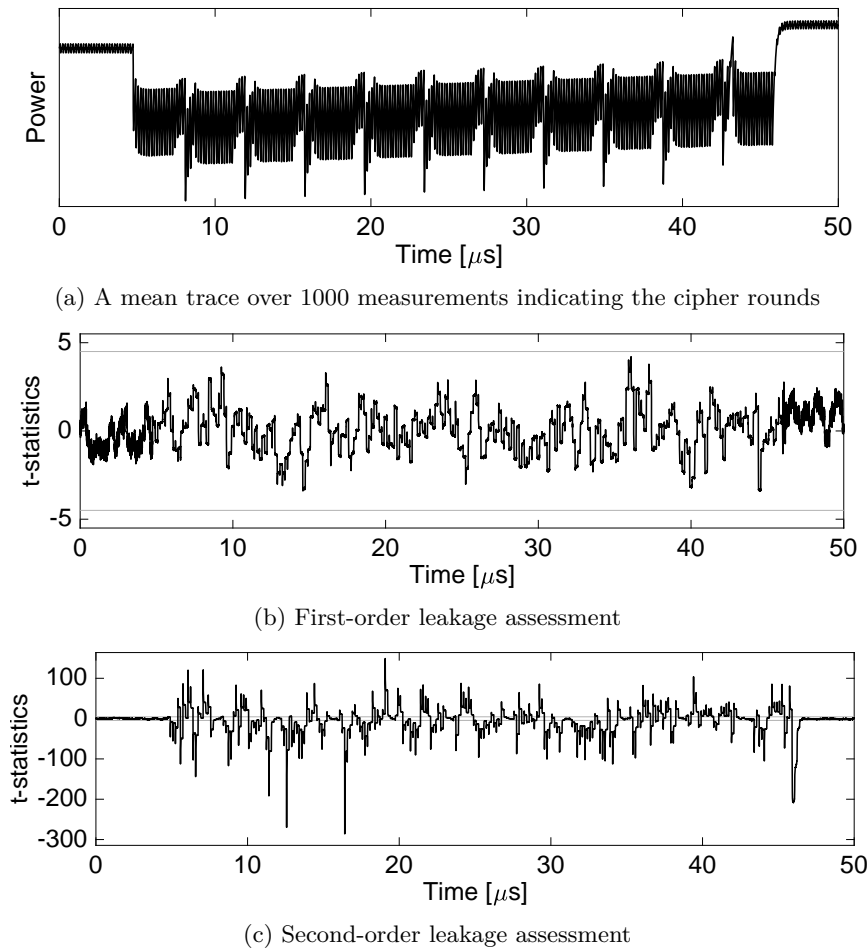


Figure 5: Experimental analysis of our first-order AES byte-serial encryption design (covering the entire encryption); fixed vs. random t-test results using 100 million traces.

Table 2: Performance figures of our case studies.  
(using Synopsis Design Compiler and UMC 180 standard cell library)

Design	Scheme	Order	Random.	Area	Delay	Latency
		$d$	[bit]	[GE]	[ns]	[cycle]
<b>AES</b> Serial	unprotected	0	0	3646	8.13	195
	GHPC	1	8	86 326	21.34	215
	GHPC <sub>LL</sub>	1	2048	76 339	23.48	205
<b>PRESENT</b> Serial	unprotected	0	0	2139	5.12	545
	GHPC	1	4	5604	5.76	607
	GHPC <sub>LL</sub>	1	64	5253	5.92	576
<b>PRESENT</b> Round- based	unprotected	0	0	2798	4.39	31
	GHPC	1	72	31 559	5.22	62
	GHPC <sub>LL</sub>	1	1152	25 264	5.28	31

clock cycle to maintain the first-order security.

**Leakage Assessment.** Using a fix-versus-random Test Vector Leakage Assessment (TVLA) methodology according to [SM15], Figure 5b and Figure 5c show evaluation results for first-order and second-order statistical moments using 100 million power traces. As expected, our design does not exhibit any observable leakage for the first-order statistical moment while expectedly we could observe significant differences in the second-order statistical moment. These results indeed confirm our theoretical security evaluations for the GHPC construction, showing its applicability to arbitrary Boolean functions in order to construct generic and composable PINI-secure gadgets.

### 5.3 Nibble-Serial PRESENT

For our second case study, we implemented the nibble-serial design of Poschmann et al. [PMK<sup>+</sup>11], realizing the PRESENT encryption where a single S-box instance is shared for the entire data and key processing. Per clock cycle, both state and key registers are shifted nibble-wise to conduct key addition and S-box look-up at the same time, while the permutation layer is done in parallel (in a single clock cycle). Again, we constructed a general design, in which the user can set the desired GHPC or GHPC<sub>LL</sub> scheme. The number of required fresh masks as well as the latency of the S-box, required for the control logic, is automatically adjusted accordingly. For more detail on the design architecture, we refer to the HDL code given in the GitHub: <https://github.com/Chair-for-Security-Engineering/GHPC>. Table 2 also lists the performance figures of our designs including the area overhead, required fresh randomness, latency, and delay.

Focusing on our GHPC design, we collected 100 million traces and performed fix-versus-random TVLA at different orders. The results shown in Figure 6 confirm our claims and expectations on the security level of our construction.

### 5.4 Round-Based PRESENT

We also implemented the PRESENT encryption function in a round-based fashion. The unprotected design performs each cipher round in a single clock cycle, resulting in 31 cycles for the entire encryption. The first-order GHPC design needs 2 clock cycles per round while forming a pipeline design, i.e., encrypting two plaintexts in consecutive clock cycles, resulting in 62 clock cycles for two encryptions. Note that we made use of the internal registers of the S-box as the state register. This allowed us to keep 31 clock cycle

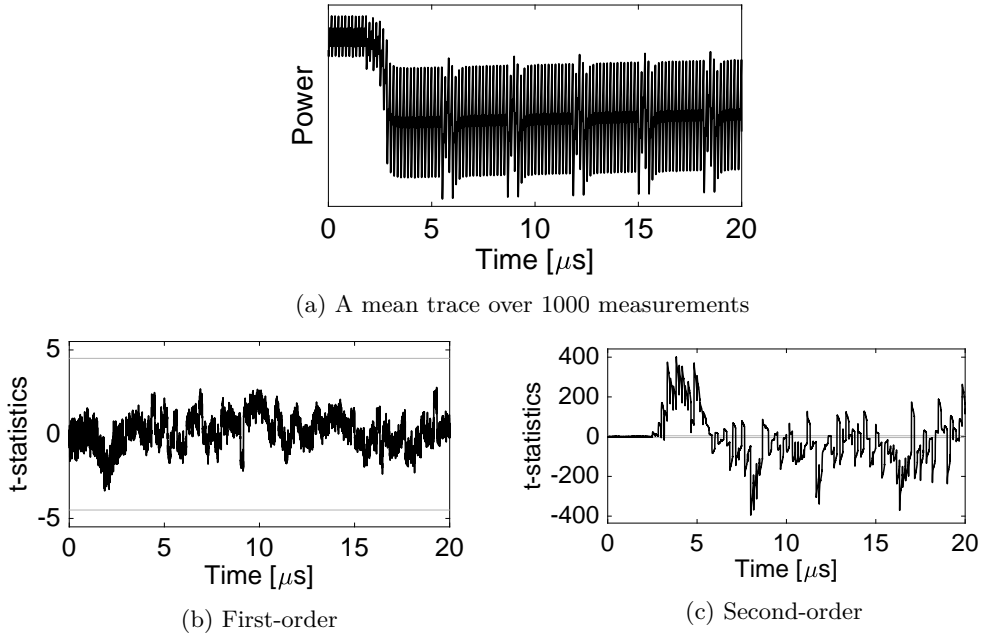


Figure 6: Experimental analysis of our first-order PRESENT nibble-serial encryption design (covering the first five rounds); fixed vs. random t-test results using 100 million traces.

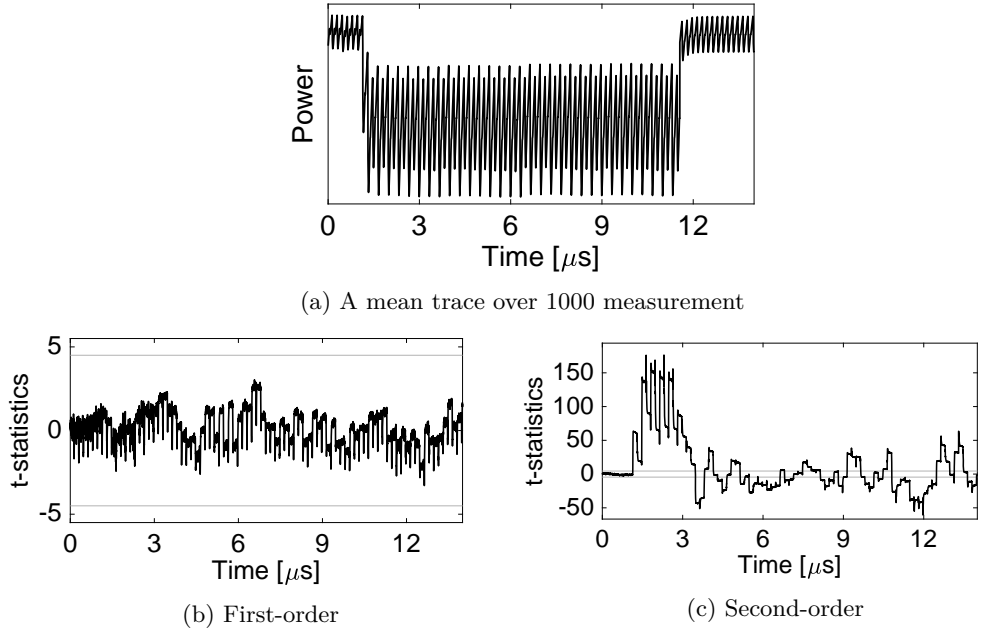


Figure 7: Experimental analysis of our first-order PRESENT round-based encryption design (covering the entire encryption); fixed vs. random t-test results using 100 million traces.

latency in  $\text{GHPC}_{\text{LL}}$  design (see Table 2). Similar to all other case studies, we practically examined this construction by performing the same leakage assessment at different orders. The results, which are along the same line as the formerly presented ones, are shown in Figure 7. Nevertheless, Table 2 covers the performance figures of this design as well.

As a remark, the evaluation results of the  $\text{GHPC}_{\text{LL}}$  circuits of all case studies are very



similar to the figures presented above. Therefore, we omit showing the identical results.

## 6 Discussions and Conclusions

In this work, we developed and presented a generic framework to construct trivially composable hardware private circuits with a compact latency from arbitrary vectorial Boolean functions. Following the concept of Shannon’s decomposition, we derived generic circuit constructions which offer both, first-order probing security in the presence of glitches and trivial composability, by fulfilling the notion of PINI in the robust probing model. More specifically, we presented the fundamental design principles, security analyses, and simple examples to illustrate our contribution.

After establishing the concept, we compared our constructions to state-of-the-art masking schemes. Based on this comparison, we conclude that our approach can be used to achieve optimized designs for different metrics, in particular focusing on latency, randomness, and area.

In terms of **latency**, our proposed  $\text{GHPC}_{\text{LL}}$  gadgets outperform all handcrafted and carefully optimized constructions, independent of the complexity and appearance of the underlying Boolean function. In fact, all our presented results constructed according to the  $\text{GHPC}_{\text{LL}}$  approach have the smallest latency of a single cycle, while still being PINI-secure, however, introducing higher demands on area and fresh randomness.

Then, considering **low-randomness** optimizations, our  $\text{GHPC}$  provides the best results in comparison to related works due to its independence on the targeted Boolean function and its appearance, while the number of random bits only depends on the number of outputs the underlying function has. Focusing on the designs with low fresh randomness, our  $\text{GHPC}$  approach scales well particularly for larger functions, e.g., an AES S-box, with only modest increase of latency in comparison to the  $\text{GHPC}_{\text{LL}}$  approach while requiring only 8 bit fresh randomness as it is a function with 8-bit output.

Eventually, **low-area** constructions for all S-boxes can be achieved by replacing the  $\text{HPC1}$  or  $\text{HPC2}$  2-input AND gadget with our proposed  $\text{GHPC}_{\text{LL-AND}}$  construction. Interestingly, such constructions provide the smallest area footprint along with competitive latency. This however comes at cost of increased demand for fresh randomness.

Furthermore, our methodology was verified by (i) testing small examples with the state-of-the-art formal verification tool SILVER [KSM20], and by (ii) experimentally evaluating several case studies. As a first case study for our first-order approach, we decided to analyze a byte-serialized version of AES where the entire S-box was translated into a single first-order secure and composable gadget based on our presented design principle. For a second case study, we constructed a secure nibble-serialized and round-based PRESENT encryption, again translating the entire S-box into a single gadget following the corresponding first-order design concepts. All case studies evidently support our theoretical findings by showing no leakage in the first-order statistical moment when performing a non-specific leakage assessment.

To conclude, our generic methodology for constructing masked gadgets for arbitrary vectorial Boolean functions pioneers automatic generation of masked circuits in hardware solely based on the function expression and with a constant latency of 2 clock cycles for  $\text{GHPC}$  and 1 clock cycle for  $\text{GHPC}_{\text{LL}}$  (at cost of higher fresh randomness). A fundamental question, which is not yet answered and still needs proper attention, is how expensive it is to generate a certain number of fresh masks per clock cycle. For this, choosing area, energy, power, or delay as the metric and cost function are certainly possible choices. However, without any detailed insight on the cost factors, we cannot easily prefer one design over another even though they have, for example, the same latency.

Since the presented solution is restricted to the first order with 2 shares, extension of the technique to cover higher-order security is naturally among our future works. As our

approach pioneers the automated construction of masked hardware circuits, development of a proper tool is an interesting exercise to pursue. In this context, exploration of trade-offs between randomness and latency for functions larger than 4 bits through clever construction of atomic gadgets is an interesting question also left open for future work.

## Acknowledgments

The work described in this paper has been supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and through the project 393207943 GreenSec.

## A Appendix

We have provided the HDL code of our case studies of Section 5 in the GitHub: <https://github.com/Chair-for-Security-Engineering/GHPC>. A “PINI\_pkg.vhd” file is given for each design, where the settings of the desired implementation can be adjusted. This includes parameters like “low\_latency” with which the gadget type (GHPC/ GHPC<sub>LL</sub>) can be selected, and “pipeline” which sets if pipeline registers should be instantiated into the designs.

We further constructed the designs in such a way that it can easily realize different Boolean functions (i.e., different S-boxes). In the “PINI\_pkg.vhd” file, the number of input bits and the output bits (via parameters “in\_size” and “out\_size”) can be adjusted, and the target Boolean function can be set as a case statement of the “PINI\_Step1.vhd” file as a look-up table. This eases the process of automatic generation of GHPC and GHPC<sub>LL</sub> gadgets of arbitrary Boolean functions.

## References

- [AIS18] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private Circuits: A Modular Approach. In *CRYPTO 2018*, volume 10993 of *Lecture Notes in Computer Science*, pages 427–455. Springer, 2018.
- [BBD<sup>+</sup>15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified Proofs of Higher-Order Masking. In *EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.
- [BBD<sup>+</sup>16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong Non-Interference and Type-Directed Higher-Order Masking. In *CCS 2016*, pages 116–129. ACM, 2016.
- [BDF<sup>+</sup>17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. In *EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–566, 2017.
- [Boo48] George Boole. The calculus of logic. 1848.
- [BP12] Joan Boyar and René Peralta. A Small Depth-16 Circuit for the AES S-Box. In *Information Security and Privacy Conference, SEC 2012*, volume 376 of *IFIP*, pages 287–298. Springer, 2012.

- [CGLS20] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Transactions on Computers*, 2020.
- [CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
- [CRB<sup>+</sup>16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with  $d+1$  Shares in Hardware. In *CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212. Springer, 2016.
- [CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Information Forensics and Security*, 15:2542–2555, 2020.
- [DBR19] Lauren De Meyer, Begül Bilgin, and Oscar Reparaz. Consolidating Security Notions in Hardware Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):119–147, 2019.
- [FGP<sup>+</sup>18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.
- [GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic Low-Latency Masking in Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):1–21, 2018.
- [GM18] Hannes Groß and Stefan Mangard. A unified masking approach. *J. Cryptogr. Eng.*, 8(2):109–124, 2018.
- [GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In *CT-RSA 2017*, volume 10159 of *LNCS*, pages 95–112. Springer, 2017.
- [ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.
- [Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
- [KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - Statistical Independence and Leakage Verification. In *ASIACRYPT 2020*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.
- [MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):256–292, 2019.
- [MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In *CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

- [MPL<sup>+</sup>11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.
- [NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schl affer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptol.*, 24(2):292–321, 2011.
- [PMK<sup>+</sup>11] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-Channel Resistant Crypto for Less than 2,300 GE. *J. Cryptology*, 24(2):322–345, 2011.
- [RBN<sup>+</sup>15] Oscar Reparaz, Beg ul Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In *CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.
- [SAK] SAKURA. Side-channel Attack User Reference Architecture. <http://satoh.cs.uec.ac.jp/SAKURA/index.html>.
- [SM15] Tobias Schneider and Amir Moradi. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In *CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.
- [Tri03] Elena Trichina. Combinational Logic Design for AES SubByte Transformation on Masked Data. *IACR Cryptol. ePrint Arch.*, 2003:236, 2003.
- [WM12] Roy Ward and Timothy C.A. Molteno. Table of Linear Feedback Shift Registers. Technical Report 2012-1, University of Otago, 2012. <http://www.physics.otago.ac.nz/reports/electronics/ETR2012-1.pdf>.

## 5.2 Composable Gadgets with Reused Fresh Masks – First-Order Probing-Secure Hardware Circuits with only 6 Fresh Masks

### Publication Data

David Knichel and Amir Moradi. Composable Gadgets with Reused Fresh Masks – First-Order Probing-Secure Hardware Circuits with only 6 Fresh Masks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(3):114–140, 2022

*The work included in this thesis is a minor revision of the original paper published in the conference journal. It contains small additions.*

*This work is reproduced here with permission. This work is licensed under a Creative Commons Attribution 4.0 International License. Copyright is held by the authors. The author of this thesis is also an author of this research paper.*



**Content.** In this work, we present COMAR, a methodology that allows for first-order automated masking with only 6 fresh random bits in total (not counting fresh randomness needed of the initial input masking), regardless of the size and complexity of the unprotected circuit. This is possible as our construction enable complete randomness reuse across gadgets. We present gadget constructions for (i) a masked variant of an AND gate, from which we can derive any other atomic non-linear gadget and (ii) a masked variant of the XOR operation. As each of the gadgets adds individual latency into the design, we introduce construction of our masked AND and XOR gadget with arbitrary (unshared) input width. While slightly increasing the randomness requirements, this way, we allow to reduce the overall latency of the design. Eventually, we integrate support for Composable Gadgets with Reused Fresh Mask (COMAR) into AGEMA.

**Contribution.** The author was involved in verifying and testing the developed gadgets and formalizing security arguments for the novel construction. Last but not least, the author significantly contributed to the writing and presentation of the publication. The author of this thesis likes to thank his co-author who is also his thesis supervisor.



# Composable Gadgets with Reused Fresh Masks

## First-Order Probing-Secure Hardware Circuits with only 6 Fresh Masks

David Knichel<sup></sup> and Amir Moradi<sup></sup>

Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany  
[firstname.lastname@rub.de](mailto:firstname.lastname@rub.de)

**Abstract.** Albeit its many benefits, masking cryptographic hardware designs has proven to be a non-trivial and error-prone task, even for experienced engineers. Masked variants of atomic logic gates, like AND or XOR – commonly referred to as gadgets – aim to facilitate the process of masking large circuits by offering free composition while sustaining the overall design’s security in the  $d$ -probing adversary model. A wide variety of research has already been conducted to (i) find formal properties a gadget must fulfill to guarantee composability and (ii) construct gadgets that fulfill these properties, while minimizing overhead requirements. In all existing composition frameworks like NI/SNI/PINI and all corresponding gadget realizations, the security argument relies on the fact that each gadget requires individual fresh randomness. Naturally, this approach leads to very high randomness requirements of the resulting composed circuit. In this work, we present *composable gadgets with reused fresh masks* (COMAR), allowing the composition of any first-order secure hardware circuit utilizing only 6 fresh masks in total. By construction, our newly presented gadgets render individual fresh randomness unnecessary, while retaining free composition and first-order security in the robust probing model. More precisely, we give an instantiation of gadgets realizing arbitrary XOR and AND gates with an arbitrary number of inputs which can be trivially extended to all basic logic gates. With these, we break the linear dependency between the number of (non-linear) gates in a circuit and the randomness requirements, hence offering the designers the possibility to highly optimize a masked circuit’s randomness requirements while keeping error susceptibility to a minimum.

**Keywords:** Side-Channel Analysis · Masking · Probing Security · Composability · COMAR

## 1 Introduction

**Masking.** The increasing amount of easily accessible devices creates a permanent surge in demand for highly effective countermeasures against physical attacks, causing Side-Channel Analysis (SCA) attacks to retain their topical relevance since its first seminal description in [Koc96, KJJ99]. Since then, a considerable amount of effort has been put into mitigating the threat originating from a wide variety of diverse side channels, ranging from timing [Koc96], power consumption [KJJ99], electromagnetic (EM) emanations [GMO01], or temperature and heat dissipation [HS13]. Among a multitude of proposed methods, *masking* has evolved to be a promising technique to achieve SCA resilience due to its solid theoretical background rooted in the concepts of secret sharing. Despite the strong effort committed to masking cryptographic primitives [ISW03, Tri03, NRS11, RBN<sup>+</sup>15, GMK17, GM18], many of the proposed schemes have proven to suffer from invalid assumption or design flaws [MMSS19], eventually compromising their practical security.

**Formal Adversary Models.** In order to prevent these bugs, researchers have started to lay focus on establishing formal adversary models which aim to accurately capturing the adversary’s capabilities and the circuit behavior in the real world, while being sufficiently abstract to enable (automated) evaluation of SCA resilience on an algorithmic level [ISW03, BDF<sup>+</sup>17, PR13, FGP<sup>+</sup>18]. As a consequence, these models, on the one hand, enable detecting security flaws in an early stage of the design process. For this, a wide variety of tools have been proposed [BGI<sup>+</sup>18, BBC<sup>+</sup>19, KSM20, BDM<sup>+</sup>20, BBD<sup>+</sup>16, BGR18, BBD<sup>+</sup>15, CGLS21]. On the other hand, they allow following a more systematic approach when constructing provable-secure masking schemes. The ISW  $d$ -probing model – initially introduced in [ISW03] – and its extension to model physical defaults contributing to data leakage in hardware implementations [FGP<sup>+</sup>18], has turned out to be an adversary model well suited for finding implementations that are provably resilient against SCA attacks due to both, its sufficiently high abstraction and the existing reduction into the *Noisy Leakage Model* [DDF14], which is considered to be the closest to reality with respect to accurately modeling leakage behavior.

**Composable Gadgets.** As it has proven to be a hard task to mask circuits realizing large functions for high security orders, a new line of research has emerged aiming to formally define composability notions which enable the construction of masked circuits in a divide-and-conquer fashion. Here, the goal is to derive masked realizations of atomic logic gates – typically AND and XOR – that fulfill the properties specified by the composability notion, and in so doing, guaranteeing security in the  $d$ -probing model, even when interconnected into a larger circuit. Hence, this reduces the task of finding (higher-order) masked versions of large circuits to constructing masked, atomic logic gates that are in conformity with the security notions and then modularly constructing the overall circuit based on these masked gates – commonly referred to as *gadgets*. In the context of the  $d$ -probing model, Non-Interference (NI)/Strong Non-Interference (SNI) [BBD<sup>+</sup>15, BBD<sup>+</sup>16] and Probe-Isolating Non-Interference (PINI) [CS20] were introduced as such notions, where SNI ironed out flaws of NI whose properties were not sufficient to guarantee composability. As the scope of SNI was originally limited to single-output gates, Cassiers et al. generalized it in [CS20], but at the same time introduced PINI as an elegant notion to construct composable gadgets which further reduces implementation overheads and allows trivial realization, i.e., share-wise application, of linear functions. Accompanied by the introduction of this wide variety of notions, many concrete gadget constructions have been proposed, either realizing small logic gates [ISW03, BDM<sup>+</sup>20, BBD<sup>+</sup>16, CS20, CGLS21, CS21], or even arbitrary logic functions [KSM22].

**Randomness Reduction.** As gadgets typically realize atomic logic functions like a simple 2-input AND gate, and fresh randomness is required for each of these gadgets, the composition to entire cipher designs results in a significant randomness overhead being introduced to the circuit. As fresh randomness needs to be provided by a source, i.e., Pseudo-Random Number Generators (PRNGs), randomness requirements directly translate into area consumption on a chip and hence into an increase in costs of production. Consequently, it is an interesting research topic to elaborate on how randomness overhead of composable gadgets can be further reduced. One possible solution is to extend the functionalities of the gadgets while preserving individual randomness requirements. In [KSM22], a methodology was presented that allows the construction of first-order-secure gadgets realizing arbitrary Boolean functions with only one random bit per output. Although this certainly leads to randomness reductions compared to the utilization of 2-input gadgets when constructing large masked circuits, it also significantly increases the area requirements of the implementation.

Another approach is to minimize the randomness requirements per gadget. In [GSM<sup>+</sup>19],



Gross et al. presented a methodology to derive a first-order-secure implementation from AND and XOR gadgets requiring two fresh random bits in total – including the initial sharing. This approach is restricted to software implementations, i.e., does not guarantee SCA resilience in the presence of physical defaults [WM18]. Another drawback is that interconnection of gadgets is not trivial, i.e., the designer has to follow a certain set of rules when composing the gadgets.

Faust et al. introduced a methodology for reusing randomness in masked implementations in [FPS17]. Interestingly, the authors proposed a scheme that requires only two fresh random bits to achieve first-order security in the probing model, and they applied their findings to an AES implementation. However, their designs do not offer free composability, as the protected implementation must be carefully constructed to suit a specific structure. Moreover, the work does not consider physical defaults that drive leakage in their discussions, making it not directly applicable to the hardware context.

Theoretical discussions on defining the necessary properties of PRNGs to reduce the overall randomness requirements of masked implementations are presented in [IKL<sup>+</sup>13, CGZ20, GIS22a, GIS22b]. In [CGZ20], Coron et al. demonstrated a practical application of their findings by protecting an AES at the second security order using only 384 random bits. However, their work does not take into account physical defaults that may occur in hardware. In delimitation to these works, our goal is to derive concrete, secure, and freely composable structures in the hardware context that utilize very few fresh random bits and are well-suited for automated masking.

**Our Contributions.** In this work, we present COMAR, a methodology for achieving free composition of hardware gadgets that enable construction of arbitrary first-order-secure hardware implementations in the  $d$ -probing model under glitches, utilizing – apart from the initial sharing – only 6 fresh random bits in total. In this context, we construct freely-composable gadgets, realizing 2-input XOR and AND gates which can be trivially converted to all atomic logic gates. We further show how to extend such gadgets to cover an arbitrary number of inputs at the cost of investing more fresh masks which can still be reused by other gadgets. We demonstrate that our constructions lead to a significant reduction with respect to randomness overhead introduced into the design compared to existing schemes, offering the same composability and security guarantees. Our work hence paves the ground for enabling the designer to optimize for different overhead parameters when masking a cryptographic implementation, while keeping design constraints and error susceptibility as low as possible. We further practically verify our findings by means of case studies and leakage assessments.

**Outline.** We first present a brief recap of all necessary concepts and methodologies in Section 2. In particular, we give an introduction to the robust  $d$ -probing model and all well-established composability notions. In Section 3, we present our novel methodology for constructing gadgets with constant randomness requirements, and argue about their first-order security and composability, before we discuss the practical implications of our work by means of different case studies in Section 4. After presenting a dedicated comparison to the state of the art in Section 5 and conducting an exemplary leakage assessment in Section 6, we conclude our work in Section 7.

## 2 Preliminaries

### 2.1 Notations

While we denote random variables by capital letters, e.g.,  $X \in \mathbb{F}_2$  is a binary random variable, we use bold capital letters, e.g.,  $\mathbf{X}$ , for sets containing random variables. The

$i$ -th input to a function is identified by  $X_i$ , while superscripts are used to denote share indices when dealing with masked functions. Consequently,  $X^i$  is the  $i$ -th share of  $X$ . Moreover,  $\mathbf{X}_i$  denotes the set of all elements in  $\mathbf{X}$  with subscript  $i$ . When masking a function  $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$  with  $t$  shares per input, the set containing all input shares is given as  $Sh(\mathbf{X}) = [X_0^0, X_0^1, \dots, X_0^{t-1}, X_1^0, \dots, X_{n-1}^{t-1}]$ . In the same manner, for a set of share indices  $\mathbf{I} \subseteq [0, \dots, t-1]$ ,  $Sh(\mathbf{X})^{\mathbf{I}}$  denotes the set of all input shares  $X_{0 \leq i < n}^{s \in \mathbf{I}}$ . Eventually, drawing a value  $X$  uniformly and at random from a set  $\mathcal{R}$  is denoted as  $X \xleftarrow{\$} \mathcal{R}$ .

## 2.2 Boolean Masking

Rooted in the concept of secret sharing, a *Boolean Masking* of a secret  $X \in \mathbb{F}_2^n$  is a set  $\mathbf{X} \in \mathbb{F}_2^{n \times s}$  of  $s$  independent secret shares  $X^i \in \mathbb{F}_2^n$ ,  $0 \leq i < s$ , such that  $X = \bigoplus_{i=0}^{s-1} X^i$ . The secret shares are commonly derived by sampling  $X^i \xleftarrow{\$} \mathbb{F}_2^n$ , for  $0 \leq i < s-1$  and deriving the remaining share as  $X^{s-1} = X \oplus \left( \bigoplus_{i=0}^{s-2} X^i \right)$ . This directly implies that, in order to achieve  $d$ -th order security, all sensitive data has to be split in at least  $d+1$  shares, i.e., two shares in the case of first-order security.

## 2.3 Circuit Model

As derived in [ISW03] and later in its extension [FGP<sup>+</sup>18], any stateful, deterministic circuit  $C$  can be modeled as a Directed Acyclic Graph (DAG)  $G_C = \{\mathcal{V}, \mathcal{E}\}$  with  $\mathcal{V}$  being the set of vertices and  $\mathcal{E}$  the set of edges of  $G_C$ . The edges represent wires carrying elements of  $\mathbb{F}_2$  while the vertices model combinational gates such as AND and XOR or memory gates, i.e., registers. On any circuit invocation, registers output the previous input to the gate while storing the current input for the next invocation. Eventually,  $G_C$  realizes a Boolean function  $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ .

**Encoded Circuit Model.** To precisely define the adversary's capabilities when probing a masked circuit, a more fine-grained breakdown of the circuit becomes necessary. As defined in [AIS18], a *circuit compiler* consists of three algorithms. The **COMPILE** algorithm is deterministic and takes a circuit  $C$  as input and outputs a randomized (masked) circuit  $\tilde{C}$ . The probabilistic **ENCODE** algorithm takes as input  $\mathbf{X}$  and outputs the encoded input  $\tilde{\mathbf{X}}$ . In our case, this corresponds to performing Boolean masking as described in Section 2.2. **DECODE** is a deterministic algorithm, eventually taking encoded data  $\tilde{\mathbf{Y}}$  and outputting  $\mathbf{Y}$ . In the case of Boolean masking, this can be simply achieved by building the XOR-sum of all shares, i.e., unsharing.

Given these three algorithms, data processing in a shared manner can be described by computing  $\mathbf{Y} \leftarrow \text{DECODE} \circ \tilde{C} \circ \text{ENCODE}(\mathbf{X})$  for  $\mathbf{Y} \leftarrow C(\mathbf{X})$ . It is important to highlight that the adversary's probing capabilities are limited to only collect observations in  $\tilde{C}$ , while the actual execution of the **ENCODE** and **DECODE** operations stay hidden. In other words, these operation are known to the adversary, i.e., no obscurity, but they cannot be probed.

## 2.4 Probing Security

### 2.4.1 $d$ -Probing Model

The traditional ISW  $d$ -probing model [ISW03] grants the attacker the ability to probe up to  $d$  wires in the circuit and hence to observe the values carried by them. Modeling the circuit behavior relies on the assumption that at any given time, each wire carries a stable signal defined by the combinatorial function contributing to it. As a result, a circuit is



## 2.6 Probe Simulatability

*Probe simulatability* helps to formalize probe propagation, i.e., dependencies between probes placed on the encoded circuit and input shares. It can be formally defined as follows.

**Definition 2** (Perfect Probe Simulation). Given a set of (extended) probes  $\mathbf{P}$  with cardinality  $|\mathbf{P}| = t$  placed on a masked circuit  $\tilde{\mathbf{C}}$ ,  $\mathbf{P}$  is said to be *perfectly simulatable* by a set of input shares  $\mathbf{S}$  iff there exist a simulator  $\text{SIM}$ , such that for any value for the inputs of  $\tilde{\mathbf{C}}$ , the joint probability distribution over  $\mathbf{P}$  and  $\text{SIM}(\mathbf{S})$  are equal, where  $\text{SIM}(\mathbf{S}) : \mathbb{F}_2^{|\mathbf{S}|} \mapsto \mathbb{F}_2^t$  with input  $\mathbf{S} \subseteq \text{Sh}(\mathbf{X})$  is a probabilistic polynomial time (p.p.t.) simulator.

With the help of this formal definition, we can give a definition for the common composability notions in the following.

### 2.6.1 Non-Interference/Strong Non-Interference

As introduced in [BBD<sup>+</sup>15], NI does not differentiate between internal and output probes and limits probe propagation likewise for both. Due to the lack of any distinction between internal and output probes placed on a masked circuit, NI has proven to be non-sufficient to guarantee composability. As a remedy, its definition was adjusted by Barthe et al. in [BBD<sup>+</sup>16], resulting in the notion of SNI where probes placed on the output of a gadget are not allowed to propagate into any input share at all.

### 2.6.2 Probe-Isolating Non-Interference

As the definition of SNI only covers single-output gadgets in its original form, Cassiers et al. extended it in [CS20] to be applicable to multiple-output gadgets as well. At the same time, they introduced the notion of PINI as an alternative to guarantee composability. Its advantages stem from its reduced overhead and enabling trivial construction, i.e., share-wise application, of masked linear functions without the need for any fresh randomness and without introducing any additional latency into the design.

Borrowed from Domain-Oriented Masking (DOM) [GMK16], share domains where introduced and probe propagation of output probes was restricted to only occur within the same share domain, while internal probes are limited to propagate within a single, but arbitrary, share domain.

**Definition 3** (*d*-Probe-Isolating Non-Interference (PINI)). Let  $\mathbf{P}_I$  be the set of internal probes with  $|\mathbf{P}_I| = t_1$ . Further, let  $\mathbf{I}_O$  be the index set (share domains) assigned to the output wires probed by  $\mathbf{P}_O$  with  $|\mathbf{I}_O| = t_2$  and  $\mathbf{P}_O$  containing all output probes.

A masked circuit  $\tilde{\mathbf{C}}$  provides *d*-Probe-Isolating Non-Interference iff for every set of probes  $\mathbf{P} = \mathbf{P}_I \cup \mathbf{P}_O$  with  $t_1 + t_2 \leq d$ , there exists a set  $\mathbf{I}_I$  of circuit input share indices with  $|\mathbf{I}_I| \leq t_1$  such that  $\mathbf{P}$  can be perfectly simulated by  $\mathbf{S} = \text{Sh}(\mathbf{X})_{\mathbf{I}_I \cup \mathbf{I}_O}$ .

## 2.7 Hardware Private Circuits

Along with the introduction of this multitude of notions, several concrete realizations of composable gadgets have been proposed. As a limitation, these gadgets mostly cover only atomic gates – like 2-input AND or XOR – as finding efficient constructions, which are provably secure in the (robust) *d*-probing model under a sufficient composability notion, is a hard task for large functions. The only exception poses GHPC [KSM22], which is a methodology for constructing first-order secure and trivial composable gadgets for arbitrary, vectorial Boolean functions requiring only one bit of fresh randomness per coordinate function, but leaves a relatively large area footprint. Nevertheless, all known

gadgets leverage the introduction of gadget-individual fresh randomness to guarantee the confirmatory to the corresponding probe propagation restrictions and hence composability. As complete cipher designs are composed of many of these fundamental gates (AND-XOR), this directly implies a significant randomness overhead for the implementation of full encryption/decryption functions. A concrete instantiation of a gadget realizing a simple AND gate and fulfilling trivial composability by means of the PINI notion is given as HPC2 which is initially presented in [CGLS21]. HPC2 contains two register stages, introducing a latency of two clock cycles regardless of the security order. In Figure 1, a schematic representing an exemplary circuit composed of first-order HPC2 AND gadgets is depicted, where each gadget is supplied with an individual fresh random bit, already resulting in a total of 7 fresh random bits for this simple example. Note that every share-wise implementation of a linear function is already PINI-conform. Hence, the XOR operation does not introduce any additional randomness or latency into the design.

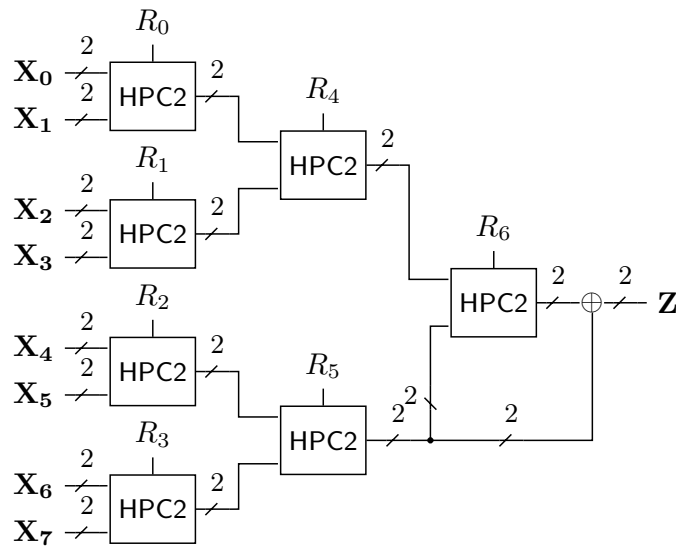


Figure 1: Example for a circuit composed of first-order HPC2 AND gadgets.

## 2.8 Automated Generation of Masked Hardware

Composable hardware gadgets are well suited for automated masking of hardware designs. The underlying methodology is to simply synthesize a given unprotected design utilizing a restricted library including only those cells for which composable gadgets exist, before substituting each of the resulting cells with the corresponding gadget. If these gadgets provably guarantee secure composition, the resulting netlist will be secure in the (robust) probing model.

AGEMA [KMMS22] is a software tool for realizing this transformation from an unprotected netlist to a protected one. For this, the Verilog netlist is represented as a graph before it is translated into a Mealy machine, i.e., a combinational circuit and a single main register stage. The designer can then specify which part of the netlist should be masked, and which family of gadgets should be instantiated. In the *naive* approach integrated in AGEMA, the structure of the given netlist stays unchanged, and only the cells are substituted by their corresponding gadgets. Of course, since these gadgets introduce additional latency, pipelining or clock gating is automatically applied to ensure the correct functionality of the circuit. This process elegantly ensures security in the glitch-extended robust probing model while being less error-prone than manually applying the masking at

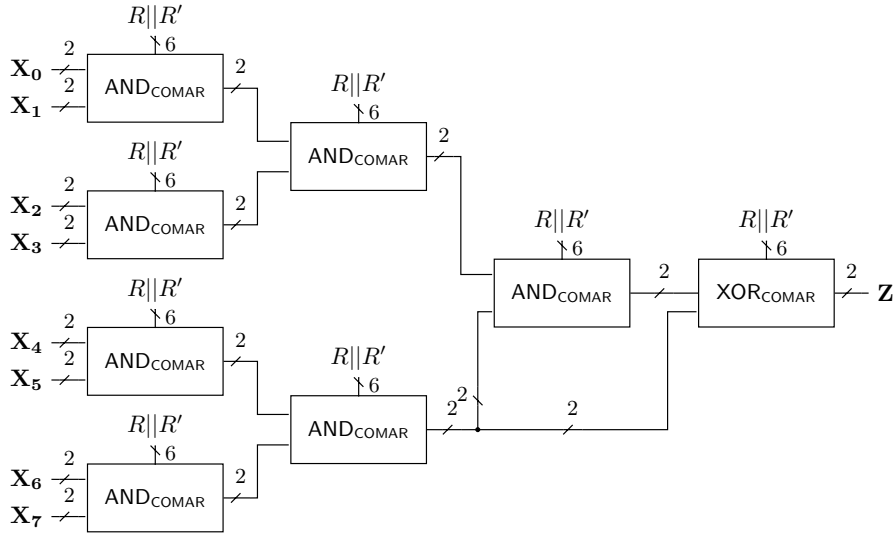


Figure 2: Exemplary circuit composed of COMAR gadgets.

the algorithmic level. In this work, we utilize AGEMA to construct the masked ciphers for our case studies, i.e., we provide our gadget definitions to AGEMA when selecting the naive approach. This intentional choice allows fair comparison with several case studies given in the original work [KMMS22].

### 3 Technique

#### 3.1 Overview

Our objective is to reduce the randomness requirements of gadget-composed cipher designs in hardware. As given in Section 2.7, each 2-input HPC2 AND gadget requires one individual fresh mask bit which is necessary to guarantee composability under the PINI notion and can thus not be nullified. Therefore, pursuing our goal, we introduce COMAR, a methodology for constructing first-order secure gadgets, which allows us to reuse the same randomness in each gadget, drastically reducing the randomness requirements compared to existing designs which rely on gadget-individual masks. Thanks to the composability of COMAR gadgets, their composition to larger circuits leads to  $d$ -probing secure designs (for  $d = 1$ ) under glitch-extended probes.

Naturally, the randomness requirements of HPC2 (and also other composable gadgets) scale with the size and complexity of the target function which is to be masked, i.e., the number of 2-input non-linear gates, while our approach only demands for 6 random bits regardless of the underlying function.

In Figure 2, the same circuit as presented in Section 2.7 is depicted where every input is split into two shares, but now the circuit is realized by instantiating our new COMAR gadgets instead of HPC2 AND. In contrast to HPC2, each of our 2-input AND gadgets needs 6 random masks, instead of only a single one, but as these fresh masks can be reused for every gadget, it is directly obvious that the break-even-point with respect to fresh randomness requirements of our approach compared to HPC2 lies at a number of 6 AND gadgets. Meaning that if a circuit contains more than 6 AND gates, our approach requires less fresh randomness compared to HPC2. This threshold is actually fulfilled for every real-world logic circuit. Figure 2 shows an example to demonstrate this break-even-point. Compared to Figure 1, our approach here needs only 6 instead of 7 bits. Naturally, this

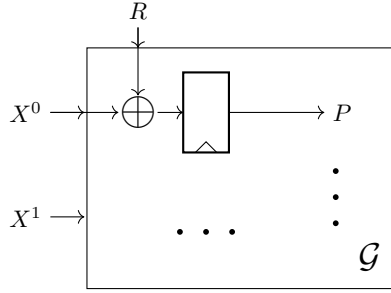


Figure 3: Example probe on a gadget.

advantage becomes more significant for larger circuit.

Note that, in contrast to PINI gadgets, a trivial, i.e., share-wise, realization of an XOR gadget is not possible in case of COMAR. Here, the  $\text{XOR}_{\text{COMAR}}$  needs 6 random mask bits, which are the same as those 6 bits reused in every other COMAR gadget. It also needs two register stages, increasing the latency requirements for composed circuits. Nonetheless, there are usecases where it is favorable for designer to trade additional latency against significantly less randomness requirements. We give more detail about this trade-off in Section 4 and Section 5.

### 3.2 On the Necessity of Introducing Fresh Randomness into Gadgets

Usually, fresh randomness is introduced into gadgets wherever probe propagation is to be stopped into certain inputs of the gadget. Often, introducing fresh randomness conveniently allows simulation of a wire independent of any other value contributing to its distribution. Considering the simple example given in Figure 3, a probe on the output of the register can be perfectly simulated tossing a fair coin if  $R$  is a fresh random bit, i.e.,  $R$  is certainly independent of any other wire in the design – in particular independent of  $X^0$ . Therefore,  $P = R \oplus X^0$  can be perfectly simulated by  $P \stackrel{\$}{\leftarrow} \mathbb{F}_2$  which is formalized by Theorem 1.

The situation changes if  $X^0$  depends on  $R$ . We assume  $X^0 = X \oplus R$  and  $X^1 = R$  as stable input signals. In this case,  $P$  is not perfectly simulatable using only  $X^0$ , as  $P = (X \oplus R) \oplus R = X$ . To simulate this, we need both input shares  $X^0$  and  $X^1$ .

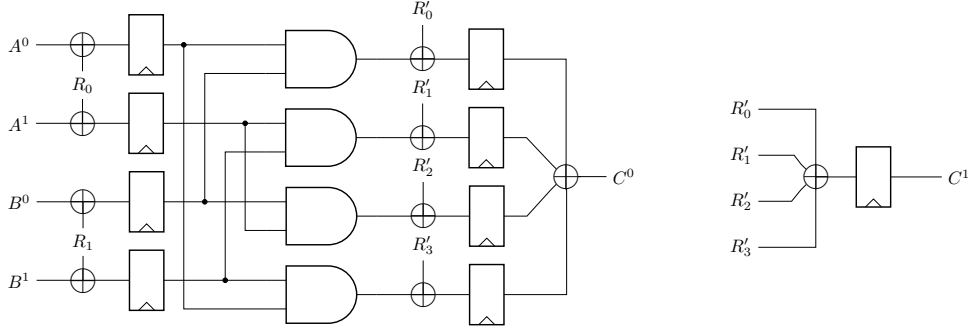
**Theorem 1.** *If a wire  $W$  of a gadget  $\mathcal{G}$  is statistically independent of  $R$ ,  $P = \text{Reg}[W \oplus R]$  can be simulated by flipping a coin, i.e.,  $P \stackrel{\$}{\leftarrow} \mathbb{F}_2$ .*

*Proof.* It holds that  $\Pr[W \oplus R = 1] = \Pr[W = 1] + \Pr[R = 1] - 2 \cdot \Pr[W = 1 \vee R = 1]$ . If now  $W$  is statistically independent of  $R$ , it follows that  $\Pr[W \oplus R = 1] = \Pr[W = 1] + \Pr[R = 1] - 2 \cdot \Pr[W = 1]\Pr[R = 1]$ . It is directly observable that if  $R$  is uniformly distributed over  $\mathbb{F}_2$ , then  $\Pr[W \oplus R = 1] = \Pr[W = 1] + \frac{1}{2} - 2 \cdot \frac{\Pr[W=1]}{2} = \frac{1}{2}$ , regardless of the value of  $\Pr[W = 1]$ .  $\square$

### 3.3 COMAR

Before presenting the details of COMAR gadgets, we define Simple Sharing below, which is required to understand the underlying concept.

**Definition 4** (Simple Sharing). A sharing  $\mathbf{F} \in \mathbb{F}_2^n$  of  $F \in \mathbb{F}_2$  is called Simple Sharing iff  $\exists! i \in [0, n-1]$  with  $F^i = F \oplus \bigoplus_{j \neq i} R_j$  and  $F^{\forall j \neq i} = R_j \stackrel{\$}{\leftarrow} \mathbb{F}_2$ , i.e.,  $n-1$  shares are set to individual random masks and one share is set to the unmasked value  $F$  added by the sum of all  $n-1$  masks.

Figure 4: COMAR first-order 2-input AND gate ( $\text{AND}_{\text{COMAR}}$ ).**Algorithm 2** COMAR first-order  $n$ -input AND ( $\text{AND}_{\text{COMAR}}^n$ )

**Input:** shares  $(A_0^0, A_0^1), \dots, (A_{n-1}^0, A_{n-1}^1)$  s.t.  $A_{0 \leq j < n} = A_j^0 \oplus A_j^1$  and fresh masks  $(R_j)_{0 \leq j < n}$  and  $(R'_i)_{0 \leq i < 2^n}$

**Output:** shares  $(C^0, C^1)$  s.t.  $C^0 \oplus C^1 = \prod_j A_j$

- 1: **for**  $\forall j \in \{0, \dots, n-1\}$  **do** ▷ input sharing refresh
- 2:    $A_j^{\prime 0} \leftarrow \text{Reg}[A_j^0 \oplus R_j]$
- 3:    $A_j^{\prime 1} \leftarrow \text{Reg}[A_j^1 \oplus R_j]$
- 4: **end for**
- 5: **for**  $\forall i \in \{0, \dots, 2^n - 1\}$  **do**
- 6:    $C'_i \leftarrow \text{Reg}\left[\prod_j A_j^{\text{bit}_j(i)} \oplus R'_i\right]$  ▷  $\text{bit}_j(i)$ :  $j$ -th bit of binary representation of  $i$
- 7: **end for**
- 8:  $C^0 \leftarrow \bigoplus_{0 \leq i < 2^n} C'_i$
- 9:  $C^1 \leftarrow \text{Reg}\left[\bigoplus_{0 \leq i < 2^n} R'_i\right]$

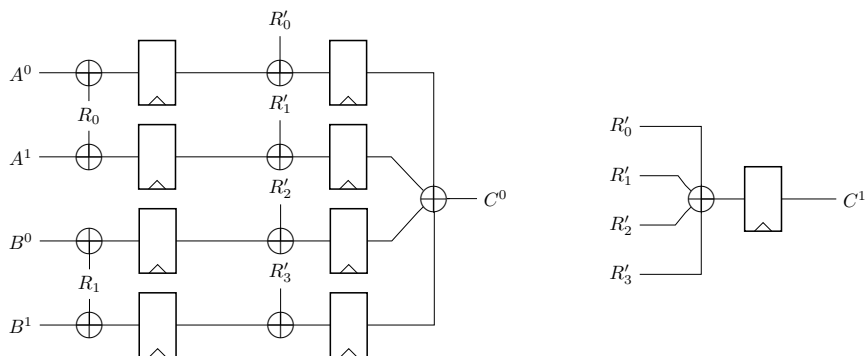
**3.3.1 AND**

Figure 4 represents the first-order COMAR 2-input AND gate. Based on the above given definitions, every signal at the output of a gate is simply shared with the same mask bit  $R$ , e.g.,  $(A^0, A^1) : (A \oplus M, M)$ , and  $(B^0, B^1) : (B \oplus M, M)$ . Therefore, each input  $A$  and  $B$  should be first refreshed using  $R_0$  and  $R_1$  respectively. We further require 4 fresh mask bits  $R'_0$  to  $R'_3$  to blind the non-linear monomials. As shown, the shared output is formed as  $(C^0, C^1) : (AB \oplus M, M)$  with  $M = R'_0 \oplus R'_1 \oplus R'_2 \oplus R'_3$ , i.e., satisfying simple sharing given in Definition 4. Further, placing any probe at the output sharing does not propagate to any input shares.

In total, the gadget has a latency of 2 clock cycles and requires 6 fresh mask bits. This is clearly larger than HPC2 2-input AND gadget (Section 2.7), but all our gadgets can receive the same fresh masks  $\langle R_0, R_1, R'_0, R'_1, R'_2, R'_3 \rangle$ . In other words, independent of the size of the circuit, and the number of instantiated 2-input AND gates, only 6 fresh masks are required.

This scheme can be extended to AND operations with a higher number of inputs. The corresponding pseudo-code is given in Algorithm 2. For an  $n$ -input AND gate,  $n$  fresh mask bits are required to refresh the sharing of  $n$  inputs, and  $2^n$  fresh mask bits for blinding the non-linear monomials. In short, in a circuit with at most  $n$ -input AND gates,  $n + 2^n$  fresh



Figure 5: COMAR first-order 2-input XOR gate ( $\text{XOR}_{\text{COMAR}}$ ).

mask bits are required which are reused by all corresponding gadgets.

Note that since the second shares  $A^1$ ,  $B^1$  and  $C^1$  in all such AND gadgets are the same, for the sake of performance, the associated gates and registers (i.e., lines 3 and 9 of Algorithm 2 and  $\text{Reg}[A^1 \oplus R_0]$ ,  $\text{Reg}[B^1 \oplus R_1]$ , and  $\text{Reg}[R'_0 \oplus \dots \oplus R'_3]$  in Figure 4) do not need to be repeated for every gadget.

### 3.3.2 XOR

We should highlight that the share-wise application of XOR operations in HPC2 naturally fulfill the composability requirements and can be easily cascaded. However, this is not the case for COMAR, since the output of every gadget is shared by the same mask, and XORing them would clearly lead to unmasked values, i.e., first-order leakage. Therefore, in contrast to HPC2, we constructed an XOR gadget, whose block diagram is shown in Figure 5 and whose structure is very similar to that of the  $\text{AND}_{\text{COMAR}}$ . Originating from another gadget's output, both inputs  $A$  and  $B$  are shared with the same  $M = R'_0 \oplus \dots \oplus R'_3$ , before they are refreshed by  $R_0$  and  $R_1$  respectively, which are the same  $R_0$  and  $R_1$  used in  $\text{AND}_{\text{COMAR}}$  gadgets. In the next stage, each of the refreshed shares is XORed with one  $R'_{i \in \{0, \dots, 3\}}$ , also being the same as those applied in  $\text{AND}_{\text{COMAR}}$  gadget. Therefore, the output shares of the XOR gadget become  $C^0 = A \oplus B \oplus M$  and  $C^1 = M$ , i.e., satisfying the definition of a Simple Sharing with the same  $M$  as used for all other inputs and outputs of other gadgets.

As a disadvantage, its area footprint is higher than the HPC2 XOR and it requires two register stages. However, similar to  $\text{AND}_{\text{COMAR}}^2$ , we can extend the construction of  $\text{XOR}_{\text{COMAR}}$  to support XORing a higher number of operands. The corresponding pseudocode is given in Algorithm 3. Similar to the  $\text{AND}_{\text{COMAR}}$ , the gates and registers associated to the second share, i.e.,  $\text{Reg}[A^1 \oplus R_0] \oplus R'_1$ ,  $\text{Reg}[B^1 \oplus R_1] \oplus R'_3$ ,  $C^1 = \text{Reg}[R'_0 \oplus \dots \oplus R'_3]$  and lines 3 and 9 in Algorithm 3 do not need to be repeated for every gadget.

### 3.3.3 Proofs

Below, we provide necessary theorems to prove the security of our constructed COMAR gadgets when composed to build larger circuits.

**Assumption 1.** *Every COMAR gadget with  $n$  inputs is supplied with two sets of fresh masks  $(R)_{0 \leq j < n}$  and  $(R')_{0 \leq i < n'}$  with  $n' = 2^n$  for  $\text{AND}_{\text{COMAR}}^n$  and  $n' = 2n$  for  $\text{XOR}_{\text{COMAR}}^n$ . For every evaluation of the gadget, each of these  $n + n'$  fresh masks is individually drawn from a uniform distribution at random and is independent of other  $n + n' - 1$  bits.*

**Theorem 2.** *If each  $R_j$  is statistically independent of  $A_j^0$  and  $A_j^1$ , for every  $0 \leq j < n$ ,  $\text{AND}_{\text{COMAR}}^n$  provides first-order security under the Probe-Isolating Non-Interference notion.*

**Algorithm 3** COMAR first-order  $n$ -input XOR ( $\text{XOR}_{\text{COMAR}}^n$ )

**Input:** shares  $(A_0^0, A_0^1), \dots, (A_{n-1}^0, A_{n-1}^1)$  s.t.  $A_{0 \leq j < n} = A_j^0 \oplus A_j^1$  and fresh masks  $(R_j)_{0 \leq j < n}$  and  $(R'_i)_{0 \leq i < 2n}$

**Output:** shares  $(C^0, C^1)$  s.t.  $C^0 \oplus C^1 = \bigoplus_j A_j$

```

1: for  $\forall j \in \{0, \dots, n-1\}$  do                                     ▷ input sharing refresh
2:    $A_j'^0 \leftarrow \text{Reg}[A_j^0 \oplus R_j]$ ,
3:    $A_j'^1 \leftarrow \text{Reg}[A_j^1 \oplus R_j]$ 
4: end for
5: for  $\forall j \in \{0, \dots, n-1\}$  do
6:    $C'_{2j} \leftarrow \text{Reg}[A_j'^0 \oplus R'_{2j}]$ ,    $C'_{2j+1} \leftarrow \text{Reg}[A_j'^1 \oplus R'_{2j+1}]$ 
7: end for
8:  $C^0 \leftarrow \bigoplus_{0 \leq i < 2n} C'_i$ 
9:  $C^1 \leftarrow \text{Reg}[\bigoplus_{0 \leq i < 2n} R'_i]$ 

```

*Proof.*

**Input to the first register stage:** A probe on the input to the first register stage observes the distribution over  $(A_j^{l \in \{0,1\}}, R_j)$ . As  $A_j^l$  is independent of  $R_j$ , this can be simulated by  $A_j^l$  and choosing  $R_j \xleftarrow{\$} \mathbb{F}_2$ . Hence, an extended probe only propagates into  $A_j^l$ .

**Input to the second register stage:** An extended probe on the input to the second register stage reveals the distribution over  $(A_0^{bit_0(i)} \oplus R_0, A_1^{bit_1(i)} \oplus R_1, \dots, A_{n-1}^{bit_{n-1}(i)} \oplus R_{n-1}, R'_i)$  for  $i \in \{0, \dots, 2^n - 1\}$ . Directly following [Theorem 1](#), each such an observation, so-called  $O$ , can be simulated by sampling  $O \xleftarrow{\$} \mathbb{F}_2^{n+1}$ .

**Output:** An extended probe on the output observes the distribution over  $(C'_0, C'_1, \dots, C'_{2^n-1})$ , with  $C'_i = \prod_j A_j^{bit_j(i)} \oplus R'_i$ . Since  $\prod_j A_j^{bit_j(i)}$  is independent of  $R'_i$ , this can be simulated by sampling the observation  $O \xleftarrow{\$} \mathbb{F}_2^{(2^n)}$ , due to [Theorem 1](#).

□

**Theorem 3.** *If each  $R_j$  is statistically independent of  $A_j^0$  and  $A_j^1$ , for every  $0 \leq j < n$ ,  $\text{XOR}_{\text{COMAR}}^n$  provides first-order security under the Probe-Isolating Non-Interference notion.*

*Proof.*

**Input to the first register stage:** A probe on the input to the first register stage is similar to that on an  $\text{AND}_{\text{COMAR}}^n$ , which is covered by [Theorem 2](#).

**Input to the second register stage:** A probe on the input to the second register stage observes the distribution over  $(A_{\lfloor i/2 \rfloor}^{bit_0(i)} \oplus R_{\lfloor i/2 \rfloor}, R'_i)$  for  $i \in \{0, \dots, 2n-1\}$ . Due to the independence of  $R_{\lfloor i/2 \rfloor}$  and  $R'_i$ , following [Theorem 1](#), each such an observation can be simulated by sampling  $O \xleftarrow{\$} \mathbb{F}_2^2$ .

**Output:** A glitch-extended probe on the output observes the distribution over  $(C'_0, C'_1, \dots, C'_{2n-1})$ , with  $C'_i = A_{\lfloor i/2 \rfloor}^{\text{bit}_0(i)} \oplus R_{\lfloor i/2 \rfloor} \oplus R'_i$ . Since each  $A_j^{l \in \{0,1\}}$  is independent of  $R_j$  (the assumption of the theorem), this can be simulated by sampling the observation  $O \xleftarrow{\$} \mathbb{F}_2^{(2n)}$ , due to Theorem 1.

□

**Assumption 2.** *In a circuit composed of only COMAR AND gadgets with at most  $n$  inputs and COMAR XOR gadgets with at most  $m$  inputs, all gadgets receive the same set of fresh masks  $(R)_{\max(n,m)}$  and  $(R')_{\max(2^n, 2m)}$ .*

**Theorem 4.** *Every masked circuit composed of COMAR gadgets is first-order secure in the glitch-extended  $d$ -probing model with  $d = 1$ .*

*Proof.*

Let  $C_{\text{COMAR}}$  be a masked (sub)-circuit with  $m$  outputs realizing the Boolean function  $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ ,  $F = \langle F_0, F_1, \dots, F_{m-1} \rangle$  with each  $F_i$  being a coordinate function  $\mathbb{F}_2^n \mapsto \mathbb{F}_2$ , solely composed of COMAR gadgets.

- $\text{AND}_{\text{COMAR}}^m \circ C_{\text{COMAR}}$ . First, we consider the case where the inputs to an  $\text{AND}_{\text{COMAR}}^m$  are the outputs of a circuit composed of  $\text{XOR}_{\text{COMAR}}$  and  $\text{AND}_{\text{COMAR}}$ , i.e., only 2-input gadgets. A probe placed on such an interconnection hence observes distributions of one of the following forms:

$$(F'_i \oplus R'_0, F''_i \oplus R'_1, F'''_i \oplus R'_2, F''''_i \oplus R'_3) \quad \text{or} \quad (R'_0 \oplus R'_1 \oplus R'_2 \oplus R'_3),$$

with  $F_i = F'_i \oplus F''_i \oplus F'''_i \oplus F''''_i$  when the output of either an  $\text{AND}_{\text{COMAR}}$  or an  $\text{XOR}_{\text{COMAR}}$  is probed. These probed signals are the input of the next  $\text{AND}_{\text{COMAR}}^m$  as well which are blinded by the corresponding  $R_i$ . Following Assumption 2, the above-given probed values are independent of  $R_i$ ; hence, their simulatability is implied by Theorem 2.

The same holds when  $m$  outputs of  $C_{\text{COMAR}}$  are provided by larger gadgets, i.e.,  $\text{AND}_{\text{COMAR}}^{n_1 > 2}$  and  $\text{XOR}_{\text{COMAR}}^{n_2 > 2}$ . More precisely, a probe placed on an output of  $C_{\text{COMAR}}$  observes distributions which are blinded by  $R'_j$  with  $0 \leq j < 2^{n_1}$  for  $\text{AND}_{\text{COMAR}}^{n_1}$  or  $0 \leq j < 2n_2$  for  $\text{XOR}_{\text{COMAR}}^{n_2}$ . Since the first stage of  $\text{AND}_{\text{COMAR}}^m$  blinds the inputs with the corresponding  $R_i$ , following Assumption 2 and Theorem 2, their simulatability is implied.

- $\text{XOR}_{\text{COMAR}}^m \circ C_{\text{COMAR}}$ . Here, we consider the case where the outputs of  $C_{\text{COMAR}}$  are given to an  $\text{XOR}_{\text{COMAR}}^m$ . Since the first stage of  $\text{XOR}_{\text{COMAR}}^m$  is identical to that of  $\text{AND}_{\text{COMAR}}^m$ , the statements given above for  $\text{AND}_{\text{COMAR}}^m \circ C_{\text{COMAR}}$  holds true here as well.

□

### 3.3.4 Other Gates

Other gadgets, e.g., NOT, NAND, OR, NOR, and XNOR, can be easily constructed. In order to maintain the simple sharing, negation should be performed on the first share. More precisely,  $(\overline{X^0}, X^1)$  should be the output of the NOT gadget. The same holds for NAND and XNOR. Since  $A|B = \overline{\overline{A}\overline{B}}$ , an OR gadget is constructed by placing a NOT gate at the first share of all inputs and the output of an AND gadget. Therefore, all statements and proofs given in Section 3.3.3 are valid for other COMAR gadgets constructed as explained above.

### 3.3.5 Discussions

If a circuit is made by only NOT and 2-input AND, NAND, OR, NOR, XOR, and XNOR gates, its first-order secure variant requires 6 fresh mask bits independent of its size and the number of gates. When larger gadgets (i.e., with a higher number of inputs) are employed, naturally the circuit needs more fresh mask bits as a trade-off between the latency and the amount of demand for fresh randomness. If the circuit instantiates non-linear COMAR gadgets with at most  $n$  inputs and linear COMAR gadgets with at most  $m$  inputs, the number of required fresh mask bits for the entire circuit is  $\max(n, m) + \max(2^n, 2m)$ . As stated, the drawback is that the XORs would introduce additional register stages compared to the equivalent first-order HPC2 circuit, whose required number of fresh mask bits equals the number of 2-input non-linear gates. Further, note that HPC2 gadgets are only available for 2-input gates and hence there is no trade-off possible at all. We like to highlight that supporting  $n$ -input gadgets is a valuable feature of COMAR, as it allows a designer to structure a circuit differently, i.e., allowing gadgets with a larger or smaller number of inputs, hence adjusting the masked variant towards the specific randomness, area and latency requirements of the use case.

## 4 Case Studies

In order to examine the performance and security of our scheme, we have considered five cases studies including the round-based implementations of the full cipher encryption functions of AES-128 [DR20], Skinny64-64 [BJK<sup>+</sup>16], CRAFT [BLMR19], LED-64 [GPPR11], and Midori-64 [BBI<sup>+</sup>15].

To this end, we have constructed HDL specifications of COMAR gadgets and utilized the recently introduced open-source tool AGEMA [KMMS22], which translates the gate-level netlist of an unprotected implementation to a masked description by exchanging the gates with their corresponding masked gadgets (see Section 2.8). In order to apply AGEMA to our case studies, we first constructed the custom library of AGEMA, where the definition of the gadgets are given, i.e., the I/O port names, number of fresh masks and number of register stages of every gadget. We further used the netlist of the unprotected cores which are given in the case studies of AGEMA available through GitHub<sup>1</sup>. Direct application of AGEMA using COMAR gadgets, would construct the masked circuits correctly, but would result in individual fresh masks being given to each gadget. However, the underlying idea of COMAR is to reuse the fresh masks, i.e., giving the same fresh masks to all gadgets. Therefore, we slightly modified the source code of AGEMA to support this feature. For the sake of simplicity and to highlight the maximum possible randomness optimization, we covered just 2-input COMAR gadgets. This implies that only 6 fresh masks are used for the entire design, which are given to all COMAR gadgets.

After applying our modified version of AGEMA on the aforementioned unprotected full cipher designs we synthesized the resulting circuits with Design Compiler and NanGate 45nm standard cell library. The corresponding results are depicted in Table 1 showing the number of required fresh mask bits, the latency cycles, the critical path delay, and the area footprint of all designs. In order to enable comparison to the state of the art, we further covered the performance figures of the same designs realized utilizing HPC2, GHPC, and GHPC<sub>LL</sub> gadgets. The last two cases are introduced in [KSM22], which – similar to COMAR – are limited to first order. They can highly reduce the latency, particularly the GHPC<sub>LL</sub> variant at cost of a relatively high demand for randomness. Note that in all case studies, we applied the masking on all inputs of the circuit, i.e., in case of encryption both

<sup>1</sup><https://github.com/Chair-for-Security-Engineering/AGEMA>

plaintext and key are masked which results in applying the masking on the key schedule of the ciphers as well.

The benefit of COMAR with respect to the number of required fresh masks is obvious. In the most advantageous case, 680 fresh mask bits are required by the round-based AES-128 HPC2 design reduces to 6 bits in COMAR. The critical path delay of the COMAR circuits is also most of the time less than the equivalent HPC2 counterparts, allowing high clock frequencies. On the downside, the added latency of COMAR circuits is significantly more than the HPC2 circuits. This is somehow expected, since – as stated in Section 3 – each  $\text{XOR}_{\text{COMAR}}$  gadget introduces two latency cycles. Therefore, in designs with a high number of cascaded XOR gates, the latency of the COMAR circuit would potentially be high. As it can be seen in Table 1, the COMAR circuits of AES-128 and LED-64, which have a strong diffusion layer made by several XORs, have a higher added latency compared to the other case studies.

For the sake of completeness, we included the performance figures of some first-order AES designs in Table 1, which have not been designed based on composable gadgets, and it is not straightforward to prove their (robust) probing security when considering the entire encryption function as a whole. Nevertheless, we give a detailed discussion on comparison with the state of the art in Section 5.

## 4.1 Area Overhead

At first glance, COMAR circuits lead to a higher area overhead, which is shown in Table 1 by pure area. However, for the sake of a fair comparison, the area requirement of a design should include the part needed to realize the sources for generating the fresh random masks. This fact is usually ignored by the state of the art, and different designs are compared ignoring the area required for generating the fresh masks (see e.g., [CRB<sup>+</sup>16, MMW18, SM21b]). The reason behind such a simplification lies in the nonexistence of a suitable cost function. More precisely, the cost (e.g., required area or energy) of generating a single-bit fresh mask updated every clock cycle is not well-known. Here, we try to include this open question into our consideration by giving an intuition for possible costs of randomness creation.

**Randomness Generation.** In order to guarantee security, i.e., simulatability of every wire independent of any secret, each fresh mask bit has to be drawn independently from a uniform distribution over  $\mathbb{F}_2$ , and the adversary should not have control over or knowledge about the fresh masks. Note that, this is not an assumption specific to our work, but a general assumption of Boolean masking. For the sake of simplicity, we can for example assume that an adversary places a probe on  $X^0(X^1 \oplus R)$ . If  $R$  is drawn from a biased distribution, for example with  $\Pr[R = 0] = 0.6$  instead of 0.5,  $X^0X^1$  will be observed 20% more often than  $X^0\overline{X^1}$ , trivially leaking information about  $X = X^0 \oplus X^1$ . Hence, violating the assumption of fresh masks being drawn from a uniform distribution can naturally cause leakage. This is why the required randomness is commonly generated by a PRNG seeded with a random initial input. The independence of the generated bits, i.e., the PRNG’s non-predictability, and the discussed uniformity requirement may be assessed following [BRS<sup>+</sup>].

Looking at the state of the art<sup>2</sup>, some works used Linear Feedback Shift Registers (LFSRs), which are randomly seeded during the device power-up. Naturally, a single individual LFSR should be employed for each required fresh mask bit, i.e., sharing an

<sup>2</sup>Excluding those which generated the fresh masks on a PC and fed the cryptographic module with all required fresh masks.

Table 1: Performance figures of first-order round-based full cipher encryption functions, excluding PRNGs.

Scheme	Fresh Random	Latency [cycle]		Critical Path	Area
	[bit/cycle]	added	full	Delay [ns]	[GE]
<b>AES-128</b>					
HPC2	680	8	99	2.04	52 597
GHPC	680	8	99	1.48	67 193
GHPC <sub>LL</sub>	2720	4	55	2.28	52 450
COMAR	6	42	473	1.23	140 214
[SBM21a] <sup>a</sup>	8		216		14 256
[SM21a] <sup>a</sup>	0		246	6.25	7 136
[Sug19] <sup>a</sup>	0 <sup>b</sup>		266		17 100
[SBHM20] <sup>c</sup>	976		10		157 500
<b>Skinny64-64</b>					
HPC2	64	4	165	0.55	6 895
GHPC	64	2	99	0.80	22 850
GHPC <sub>LL</sub>	1024	1	66	0.85	18 705
COMAR	6	22	759	0.58	22 090
<b>CRAFT</b>					
HPC2	256	8	288	0.94	15 680
GHPC	64	2	96	0.75	22 106
GHPC <sub>LL</sub>	1024	1	64	0.81	15 748
COMAR	6	14	480	0.61	23 369
<b>LED-64</b>					
HPC2	64	4	165	1.98	7 691
GHPC	64	2	99	1.58	22 904
GHPC <sub>LL</sub>	1024	1	66	1.84	17 382
COMAR	6	42	1419	0.93	31 163
<b>Midori-64</b>					
HPC2	256	8	153	1.10	17 801
GHPC	64	2	51	1.05	23 901
GHPC <sub>LL</sub>	1024	1	34	1.08	19 493
COMAR	6	16	289	0.80	36 580

<sup>a</sup> Based on a byte-serial design architecture, and not using NanGate 45nm library.

<sup>b</sup> Using changing of the guards.

<sup>c</sup> Using a masked dual-rail pre-charge logic, and not based on NanGate 45nm library.

LFSR between multiple parts of the circuit would potentially lead to weaknesses, and hence security degradation, as the underlying masking schemes commonly suppose the independence of fresh masks given to different gadgets (in contrary to the core idea of COMAR). As an example, 31-bit LFSRs with feedback polynomial  $x^{31} + x^{28} + 1$  are used in multiple works [KMMS22, SM21b, MMW18, KSM22] as each one can highly efficiently be realized in FPGAs. Each such an LFSR costs 286 GE when synthesized for ASIC using

Table 2: Performance figures of different PRNGs.

Variant		Bitrate*	Delay	Area
		[bit/cycle]	[ns]	[GE]
LFSR 31-bit		1	0.17	286
LFSR 64-bit		1	0.18	565
Keccak- $f[25]$	round-based	15/12	0.61	360
Keccak- $f[25]$	unrolled	15	7.10	4.370
Keccak- $f[25]$	unrolled pipeline	15	0.62	4.320
Keccak- $f[50]$	round-based	30/14	0.74	1.154
Keccak- $f[50]$	unrolled	30	9.97	9.136
Keccak- $f[50]$	unrolled pipeline	30	0.76	16.156
Keccak- $f[100]$	round-based	60/16	1.11	2.137
Keccak- $f[100]$	unrolled	60	17.31	20.692
Keccak- $f[100]$	unrolled pipeline	60	1.12	34.192
Keccak- $f[200]$	round-based	136/18	1.14	4.173
Keccak- $f[200]$	unrolled	136	19.84	43.714
Keccak- $f[200]$	unrolled pipeline	136	1.16	75.114
Keccak- $f[400]$	round-based	336/20	1.05	7.989
Keccak- $f[400]$	unrolled	336	20.43	98.846
Keccak- $f[400]$	unrolled pipeline	336	1.06	159.780
Keccak- $f[800]$	round-based	736/22	1.11	16.209
Keccak- $f[800]$	unrolled	736	23.79	205.250
Keccak- $f[800]$	unrolled pipeline	736	1.13	356.598
Keccak- $f[1600]$	round-based	1536/24	1.04	31.361
Keccak- $f[1600]$	unrolled	1536	24.27	466.682
Keccak- $f[1600]$	unrolled pipeline	1536	1.06	752.664

\* For all Keccak variants we considered  $c = \min(0.4 \times \text{state size}, 64)$ , providing a resistance of at most  $2^c$  against state recovery.

NanGate 45nm library. If a larger period is desired, one can instantiate 64-bit LFSRs with feedback polynomial  $x^{64} + x^{63} + x^{61} + x^{60} + 1$ . This translates to 565 GE. It should be noted that the majority of the area required for an LFSR is due to the registers. The combinational circuit is made by a few XORs, and the delay of the circuit is at minimum, i.e., that of 2 or 3 XORs and the setup and hold time of registers.

Some other works like [CRB<sup>+</sup>16] used a reduced-round version of a cipher e.g., PRINCE [BCG<sup>+</sup>12] in Output FeedBack (OFB) mode implemented in an unrolled fashion. Since no detailed information about the number of covered rounds and the security of such random numbers is given, we cannot predict the corresponding area requirements. Alternatively, Sponge-Based PRNGs are known as a suitable candidate passing the statistical tests proposed by NIST [BDPA10].

Naturally, different variants of Keccak are suggested to be used in Sponge-Based PRNGs, depending on the number of required random bits and the desired period. Such designs allow skimming some bits as random values through a squeezing process, i.e., after applying the Keccak permutation function. However, the permutation function is made by

a couple of Keccak round functions depending on the size of the employed variant, i.e., the state size. For example, Keccak[ $r = 96, c = 104$ ] with a 200-bit state and a bitrate of 96 bits, which provides a resistance of about  $2^{104}$  against state-recovery attacks<sup>3</sup>, uses Keccak- $f$ [200] as the permutation function involving 18 rounds of the Keccak round function. This means that a typical round-based implementation of such a PRNG would provide 96 random bits every 18 clock cycles, which are obviously not suitable to be used as fresh masks that should be updated every clock cycle. One solution is to realize the unrolled version of such a design, i.e., one clock cycle to fully apply Keccak- $f$ [200] on the state. This highly increases the area requirements as well as the delay of the entire design. To mitigate the delay, pipeline registers can be added at each unrolled round function, realizing an unrolled pipeline design. To give an overview of the performance of such PRNGs, we constructed and synthesized different variants with different design architectures whose results are shown in Table 2. We would like to refer to [SBHM20], where a PRNG based on Keccak- $f$  is used. The authors did not fully explain which Keccak variant has been integrated, but based on the given statements, their PRNG had an internal state of 650-1050 bits, they were able to fetch up to 976 bits every clock cycle, and the PRNG (requiring 14.8 GE) did not impact critical path in their design. It is not fully clear, but we guess that at every clock cycle only one Keccak round function has been applied, not a full Keccak- $f$ .

Finally, we considered the area required for the generation of mask bits by each full-cipher HPC2, GHPC, GHPC<sub>LL</sub>, and COMAR design, and listed their area footprint for different choices of PRNGs in Table 3. This clearly shows that COMAR circuits outperform other gadget-based designs with respect to the area overhead with only one exception, i.e., when using 31-bit LFSRs, LED-64 HPC2 design has the smallest area footprint. As stated, this is due to the extensive number of XORs in the diffusion layer of LED-64, for which several XOR<sub>COMAR</sub> gadgets should be instantiated.

## 5 Comparison with the State of the Art

### 5.1 Algorithmic-Level Masking

There exists a variety of works – examples being [SM21a, SBM21a, Sug19] – aiming to achieve high optimization when masking a certain design architecture of a particular cipher like AES. We in the following refer to these approaches as *algorithmic-level* masking. The general advantage of these approaches is that they usually result in implementations with globally optimized overhead requirements. However, as disadvantages, they are bound to a certain architecture of a cipher and transferring the approach to other designs is not trivially possible or not possible at all. Another major drawback of these schemes is that they are often based on ad-hoc engineering and that there exist no formal security and composability proofs of the final design in a whole. This is due to the fact that existing verification tools like SILVER [KSM20] cannot cope with large hardware circuits. These works usually present experimental evaluations indicating the security of the design. However, their (possible) non-conformity with the formal notions in the probing security model may result in insecure implementations when experienced on different hardware platform or with more accurate measurement setup.

In [SM21a], the authors present a technique to mask a 2-input AND gate where no fresh randomness is needed. They further applied the same technique on S-boxes for different ciphers. However, we would like to highlight that the presented AND realization is not

<sup>3</sup>Note that LFSRs do not provide security against state recovery. Knowing  $l$  consecutive outputs of an LFSR with an  $l$ -bit state would lead to full recovery of the shift register. Nevertheless, it is still unknown whether security against state recovery in the context of SCA security under probing security model is required.



Table 3: Area footprint of first-order round-based full cipher encryption functions, including PRNGs.

Scheme	Area [GE]		
	LFSR 31-bit	LFSR 64-bit	Keccak, Variant
<b>AES-128</b>			
HPC2	247 281	437 205	409 195, [800]
GHPC	261 673	451 393	423 791, [800]
GHPC <sub>LL</sub>	830 370	1 589 250	1 557 778, [1600]×2
COMAR	141 932	143 608	144 534, [25]
<b>Skinny64-64</b>			
HPC2	25 218	43 093	82 009, [200]
GHPC	41 154	59 010	97 964, [200]
GHPC <sub>LL</sub>	311 569	597 265	771 369, [1600]
COMAR	23 808	25 483	26 410, [25]
<b>CRAFT</b>			
HPC2	88 973	160 473	175 460, [400]
GHPC	40 410	58 266	97 220, [200]
GHPC <sub>LL</sub>	308 612	594 308	768 412, [1600]
COMAR	25 087	26 762	27 689, [25]
<b>LED-64</b>			
HPC2	26 014	43 889	82 805, [200]
GHPC	41 208	59 064	98 018, [200]
GHPC <sub>LL</sub>	310 246	595 942	770 046, [1600]
COMAR	32 881	34 557	35 483, [25]
<b>Midori-64</b>			
HPC2	91 094	162 595	177 581, [400]
GHPC	42 205	60 061	99 015, [200]
GHPC <sub>LL</sub>	312 357	598 053	772 157, [1600]
COMAR	38 298	39 974	49 900, [25]

directly composable, and hence special care has to be taken when constructing larger circuits. This manual process might be error-prone and is not well-suited for automated masking of arbitrary implementations. In short such the masked AND gate of [SM21a] is first-order probing secure, but it cannot be used as a gadget in composed circuits. Below we give a counterexample.

Let us assume the target circuit should compute  $D \leftarrow \overline{B} | (A \& B)$ , with  $|$  and  $\&$  denoting the OR and AND operations respectively. This can be realized through computing  $C \leftarrow \text{AND}(A, B)$  and  $D \leftarrow \overline{\text{AND}(B, \overline{C})}$  by utilizing the AND gadgets of [SM21a]. Note that inversion of a masked value can be realized by inverting one of its shares, i.e.,  $\overline{C} = (C^0, \overline{C^1})$  or  $\overline{C} = (\overline{C^0}, C^1)$ . Following the authors' definition in [SM21a], we get

$$\begin{aligned}
 C^0 &\leftarrow [[\overline{A^0}B^0 \oplus B^0] \oplus [\overline{A^0}B^1]], & C^1 &\leftarrow [[A^1B^0] \oplus [A^1B^1 \oplus B^1]], \\
 D^0 &\leftarrow [[\overline{B^0}C^0 \oplus C^0] \oplus [\overline{B^0}C^1]], & D^1 &\leftarrow [[B^1C^0] \oplus [B^1C^1 \oplus \overline{C^1}]],
 \end{aligned}$$

where square brackets denote values stored into registers. Placing a single probe at the first part of  $D^0$ , i.e., the output of the register storing  $[\overline{B^0}C^0 \oplus C^0]$ , leads to

$$\begin{aligned} \overline{B^0}C^0 \oplus C^0 &= B^0C^0 = B^0((\overline{A^0}B^0 \oplus B^0) \oplus (\overline{A^0}B^1)) = \overline{A^0}B^0 \oplus B^0 \oplus \overline{A^0}B^0B^1 \\ &= A^0B^0 \oplus \overline{A^0}B^0B^1 = P. \end{aligned}$$

This leaks information about  $B$ , since if  $B = 1$ , i.e.,  $(B^0, B^1) = (0, 1)$  or  $(B^0, B^1) = (1, 0)$ , there is only one fulfilling assignment to  $A^0B^0 \oplus \overline{A^0}B^0B^1 = 1$ , namely  $(A^0, B^0, B^1) = (1, 1, 0)$ , hence  $Pr[P = 1 | B = 1] = 1/4$  instead of  $1/2$ . We verified this by SILVER [KSM20]. Note that this is an arithmetic issue not originating from glitches; hence, placing extra registers in the circuit would not have any effect on this vulnerability.

Furthermore, no formal security proof has been given in [SM21a] for the robust probing security of the full cipher implementations. Although several experimental leakage assessments were performed, we would like to highlight that those experimental assessments only allow to make security statements under a specific setup, whereas the probing security model abstracts from a certain setup in order to attest a design’s general SCA resilience.

In another work [SBM21a], the authors efficiently applied two-share Boolean masking on the cubic bijections of the decomposed AES S-box. Their design requires 16 fresh mask bits per S-box, which can be reduced to 8 bits via pipelining. Their aim was to fit such masked cubic functions into BRAM of FPGAs, although they gave ASIC performance results of a byte-serial design in the eprint version of the paper [SBM21b]. Similar to [SM21a], the security of the S-Box has been examined by SILVER, but no statement about the security of the encryption function in a whole can be provided. This becomes more relevant in serialized architectures, where various parts of the circuit are active in different clock cycles and several multiplexers decide which modules’ output should be stored in which registers. Hence, there are more locations, where the designer may unintentionally violate the probing security requirements.

Other algorithmic-level approaches, like the one presented in [Dae17], can also lead to good results. The technique, so-called changing of the guards, helps to provide uniformity in  $td + 1$  Threshold Implementations when the masked S-Box does not have a uniform output sharing. The underlying concept is conceptually very different to gate-level masking and composable security. Hence, it cannot be directly compared to our approach as it does not offer integration into automated masking tools, but need careful engineering when used in entire cipher designs. For example, we refer to [Sug19], where the same technique has been carefully applied on each module of a tower-field representation of the AES S-Box to overcome their non-uniform output sharing. Similar to many other works, the robust probing security of such a design has not been yet proven, e.g., by means of any tools.

Another promising work is the low-latency masked AES presented in [SBHM20], where the underlying concept is based a masked dual-rail pre-charge logic, called LMDPL. This allowed the authors to make large combinational circuits by LMDPL gates without instantiating register between the gates, hence leading to low-latency masked circuits. The scheme can be seen as a gate-level approach, but the application of LMDPL gates in a circuit requires specific pre-computed values which should be generated by a dedicated module called “mask table generator” designed based on the algorithm of the targeted circuit. The authors have constructed a round-based implementation, and examined its resistance by experimental evaluations showing first-order leakage after 400 million traces. As elaborated in Section 4.1, a form of Keccak –  $f$  has been used as the PRNG to generate fresh masks required for the mask table generator module. As the authors themselves stated, the technique might not exhibit the same level of resistance if it is used in algorithms and designs with smaller and/or fewer S-boxes.

## 5.2 Gate-Level Masking

*Gate-level* masking approaches, like in this work, on the other hand, deal with constructing masked variants of composable subcircuits which guarantee the security of the final implementation as a whole and under the (robust) probing security model, regardless of the underlying cipher or the design architecture. These approaches are well suited for automated masking of any unprotected implementation [KMMS22], while usually coming at the cost of higher overheads (area and/or latency). However, their benefits compared to algorithmically-masked, manually-crafted and optimized designs are (i) the ability to prove the security of the resulting circuit, and (ii) the possibility for any engineer to use existing tools in order to construct secure masked circuits without requiring extensive expertise, as the tools and composable gadgets automatically mask any given netlist, preventing engineering flaws which may possibly compromise an implementation’s practical security.

Next to NI/SNI [BBD<sup>+</sup>15, BBD<sup>+</sup>16], PINI [CS20] was introduced as a formal notion to guarantee gadget composability. As linear operation can be trivially realized in the PINI framework and the PINI-secure HPC2 AND gadget [CGLS21] introduces less overhead than existing SNI-gadgets, utilizing HPC2 gadgets is currently the most efficient way for gadget-level masking in hardware when only considering basic 2-input logic gates. A PINI-based approach for transforming any vectorial function into a first-order secure and trivial composable gadget is proposed in [KSM22]. However this approach comes with a high area demand if the considered function is large.

The main disadvantage of all NI/SNI/PINI-gadgets is that each gadget of a composed circuit requires individual and fresh randomness, naturally increasing overhead size with circuit complexity. COMAR decouples this relation by enabling the reuse of the same 6 random mask bits for every gadget in the circuit. This way, every implementation can be automatically transformed into a masked variant using only 6 individual masks in total, drastically reducing the overall randomness requirement of the resulting top-module circuit compared to other gate-level masking schemes. Of course, as for COMAR, linear operations introduce additional register stages, applying COMAR gadgets will result in a higher number of clock cycles needed for the masked implementation.

We further like to highlight that we present AND and XOR COMAR gadgets for an arbitrary number of inputs, whereas HPC2 is restricted to the realization of only 2-input non-linear gates. Hence, COMAR can be beneficial if the given unprotected implementation is optimized for 3-bit, 4-bit, or even larger gates. Although the majority of the currently available S-box implementations of different ciphers are optimized for 2-input non-linear gates, there is emerging research in this direction [BDK<sup>+</sup>21].

With COMAR, we do not aim for an overall better solution, but for offering designers an alternative when area overhead (which directly translates into production cost) is to be reduced while additional latency is acceptable.

## 6 Analyses

As the first analysis step, we examined COMAR gadgets and S-boxes of the case studies given in Section 4 by SILVER, an open-source tool verifying masked hardware circuits under different security notions, including the glitch-extended probing model. Since our work is limited to the first security order, the evaluation runtime of SILVER is rather small for our cases. Therefore, we could even examine large circuits, e.g., 4 AES S-boxes followed by a MixColumns composed of COMAR gadgets. Supporting the theory and proofs given in Section 3, SILVER verified first-order security of all our constructions. Since evaluation of larger circuits, particularly those with a sequential loop (as in our case studies), is not feasible with SILVER, we have conducted FPGA-based experiments given as follows. Note, that these experiments, in contrast to other algorithmic-level masking approaches

(see Section 5.1), are given for the purpose of presenting a complete work while security in the probing model is already guaranteed due to the gadget composability proven in Section 3.3.3.

**Setup.** For this experimental analysis, we consider our COMAR AES-128 round-based encryption function, given in Section 4, where both plaintext and key are masked. For the generation of the fresh masks we instantiated 6 individual 31-bit LFSRs as given in Section 4.1. We implemented the design on the target FPGA of a SAKURA-G SCA-evaluation board [SAK], i.e., a Xilinx Spartan-6 FPGA. We have further collected the corresponding power consumption traces by measuring the voltage drop of a shunt resistor placed in the VDD path of the target FPGA, amplified by 10 dB and then sampled by a digital oscilloscope at a sampling rate of 500 MS/s while the targeted AES design was being operated by a stable clock source at a frequency of 6 MHz. As given in Table 1, the full encryption of the AES design takes 473 clock cycles, resulting in relatively long traces to cover the entire encryption process (see Figure 6a).

As the evaluation technique, we applied the common and well-known fixed versus random t-test [CDMG<sup>+</sup>13], for which the power consumption traces are measured while the circuit is supplied with either a fixed (but masked) plaintext or a random one while the key (also masked) is kept constant for all measurements. In order to avoid false positive/negative evaluations, we followed the procedure given in [SM15] and collected 100 million traces. The analysis results are depicted in Figure 6, confirming our expectations, i.e., first-order security of the design but not second-order. Following [SM15], the second-order t-test was performed by evaluating the distribution over each sample point individually, i.e., univariate, and selecting the second central moment (variance) as the distinguisher between the fixed and random distributions. As we already identified significant leakage for the univariate second-order case, we did not extend our evaluation to multivariate analysis, i.e., the distribution over combination of different sample points.

We should highlight that the underlying cryptographic algorithm of the case study would not have an affect on the result of this evaluation. This is because – as given in Section 4 – AGEMA considers the given design as a Mealy machine, and replaces the gates with the given gadgets. At the end, the equivalent circuit with COMAR gadgets is constructed. In other words, examining other case studies presented in Section 4 would have led to the same analysis report. We additionally have examined our COMAR Skinny64-64 design, whose results are omitted for the sake of brevity. The HDL of our designs including the specification of the COMAR gadgets and the case studies of Section 4 can be found in the GitHub: <https://github.com/Chair-for-Security-Engineering/COMAR>.

## 7 Conclusions

In this work, we presented a new set of gadgets – COMAR – offering security and free composability in the glitch-extended robust  $d$ -probing model for  $d = 1$  and requiring only 6 fresh random bits in total to mask any circuit in its entirety, whereas comparable related works always introduce a linear dependency between the number of (non-linear) gates, i.e., circuit size, and the randomness requirements. With a constant number of 6 fresh random bits when utilizing 2-input COMAR gadgets, we enable the designer to highly optimize for randomness, while the achieved, free composability keeps the design-error susceptibility minimized. Moreover, we extended our gadget definitions to realize masked variants of multiple-input gates, giving the designer the opportunity to adjust a masked circuit’s latency and randomness requirements to a specific use case. We further gave formal arguments with respect to the composability and security of our constructions in the first-order (robust) probing security model and practically confirmed our findings by means of an exemplary leakage assessment. Eventually, we discussed our results on the

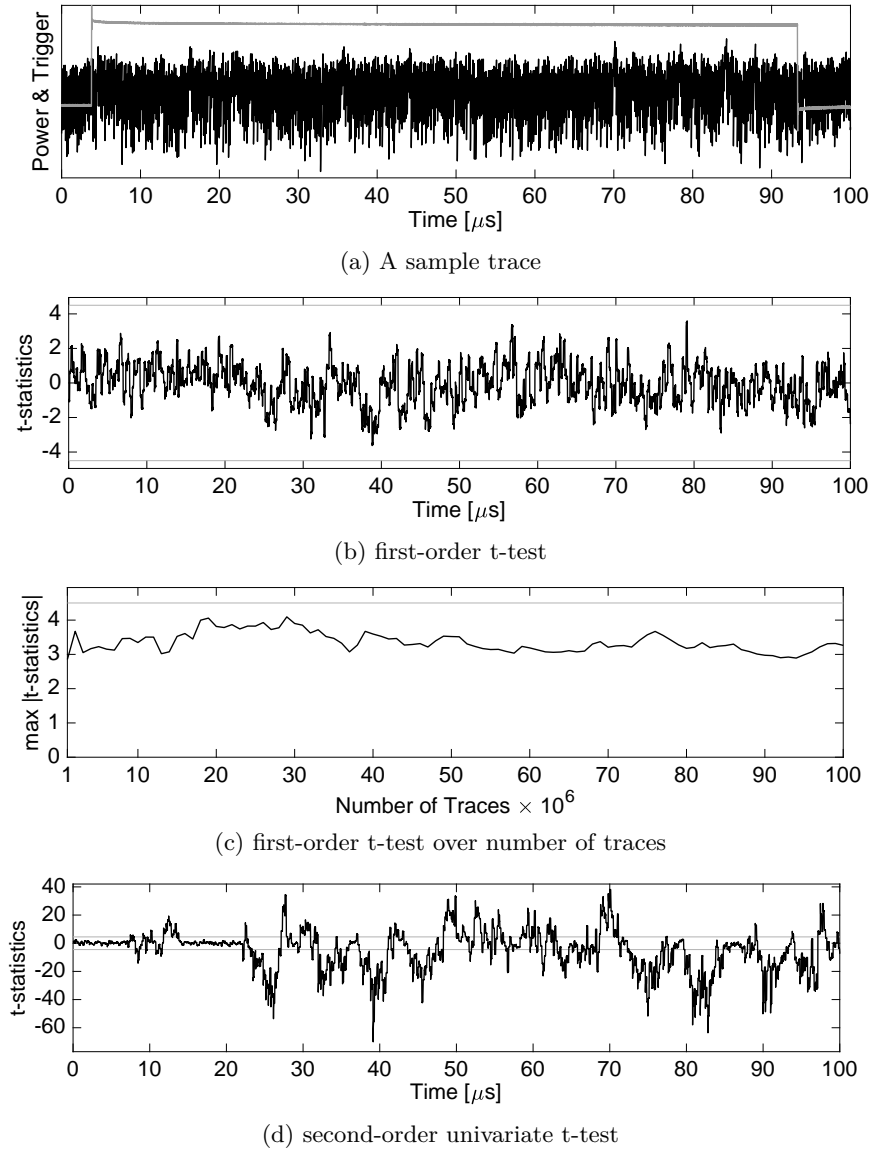


Figure 6: FPGA-based analysis of first-order round-based AES-128 encryption design, composed of COMAR gadgets, using 100 million traces.

basis of several case studies. Although our approach exceeds latency requirements when compared to related approaches, we show that our methodology outperforms them with respect to the area footprint when a fair comparison is made, i.e., when further considering the area overhead introduced by the randomness source.

To conclude, we think that COMAR offers a valuable increase in the designer’s flexibility with respect to different design metrics and use cases, while it remains an interesting question whether a randomness optimization technique can also be found for higher-order composable gadgets. Unfortunately, the same concept cannot be easily transferred to higher orders. When more than one probe is allowed, and two gadgets which reuse the same fresh masks have simple output sharing (defined in [Definition 4](#)), two probes are also enough to reveal the XOR difference between unmasked output of the corresponding gadgets. However, there might be other possible techniques to partially reuse some fresh

masks in certain circuits, which certainly deserve more attention and more research effort.

## Acknowledgments

The work described in this paper has been supported in part by the German Research Foundation (DFG) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, and through the project 393207943 "Security for Internet of Things with Low Energy and Low Power Consumption (GreenSec)".

## References

- [AIS18] Prabhajan Ananth, Yuval Ishai, and Amit Sahai. Private Circuits: A Modular Approach. In *CRYPTO 2018*, volume 10993 of *Lecture Notes in Computer Science*, pages 427–455. Springer, 2018.
- [BBC<sup>+</sup>19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In *ESORICS 2019*, volume 11735 of *Lecture Notes in Computer Science*, pages 300–318. Springer, 2019.
- [BBD<sup>+</sup>15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified Proofs of Higher-Order Masking. In *EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.
- [BBD<sup>+</sup>16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong Non-Interference and Type-Directed Higher-Order Masking. In *CCS 2016*, pages 116–129. ACM, 2016.
- [BBI<sup>+</sup>15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibusaki, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In *ASIACRYPT 2015*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.
- [BBP<sup>+</sup>16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness Complexity of Private Circuits for Multiplication. In *EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648. Springer, 2016.
- [BCG<sup>+</sup>12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventsislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In *ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.
- [BDF<sup>+</sup>17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. In *EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–566, 2017.

- [BDK<sup>+</sup>21] Anubhab Baksì, Vishnu Asutosh Dasu, Banashri Karmakar, Anupam Chattopadhyay, and Takanori Isobe. Three Input Exclusive-OR Gate Support for Boyar-Peralta’s Algorithm. In *INDOCRYPT 2021*, volume 13143 of *Lecture Notes in Computer Science*, pages 141–158. Springer, 2021.
- [BDM<sup>+</sup>20] Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. Tornado: Automatic Generation of Probing-Secure Masked Bitsliced Implementations. In *EUROCRYPT 2020*, volume 12107 of *Lecture Notes in Computer Science*, pages 311–341. Springer, 2020.
- [BDPA10] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge-Based Pseudo-Random Number Generators. In *CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2010.
- [BGI<sup>+</sup>18] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal Verification of Masked Hardware Implementations in the Presence of Glitches. In *EUROCRYPT 2018*, volume 10821 of *Lecture Notes in Computer Science*, pages 321–353. Springer, 2018.
- [BGR18] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight Private Circuits: Achieving Probing Security with the Least Refreshing. In *ASIACRYPT 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 343–372. Springer, 2018.
- [BJK<sup>+</sup>16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In *CRYPTO 2016*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.
- [BLMR19] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks. *IACR Trans. Symmetric Cryptol.*, 2019(1):5–45, 2019.
- [BRS<sup>+</sup>] Lawrence Bassham, Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Stefan Leigh, M Levenson, M Vangel, Nathanael Heckert, and D Banks. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD.
- [CDMG<sup>+</sup>13] Jeremy Cooper, Elke De Mulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, Pankaj Rohatgi, et al. Test vector leakage assessment (TVLA) methodology in practice. In *International Cryptographic Module Conference*, volume 20, 2013.
- [CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.
- [CGZ20] Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. Side-Channel Masking with Pseudo-Random Generator. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 342–375. Springer, 2020.

- [CRB<sup>+</sup>16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with  $d+1$  Shares in Hardware. In *CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212. Springer, 2016.
- [CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Information Forensics and Security*, 15:2542–2555, 2020.
- [CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):136–158, 2021.
- [Dae17] Joan Daemen. Changing of the Guards: A Simple and Efficient Method for Achieving Uniformity in Threshold Sharing. In *CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 137–153. Springer, 2017.
- [DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In *EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440. Springer, 2014.
- [DR20] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition*. Information Security and Cryptography. Springer, 2020.
- [FGP<sup>+</sup>18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.
- [FPS17] Sebastian Faust, Clara Paglialonga, and Tobias Schneider. Amortizing randomness complexity in private circuits. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 781–810. Springer, 2017.
- [GIS22a] Vipul Goyal, Yuval Ishai, and Yifan Song. Private Circuits with Quasilinear Randomness. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 192–221. Springer, 2022.
- [GIS22b] Vipul Goyal, Yuval Ishai, and Yifan Song. Tight Bounds on the Randomness Complexity of Secure Multiparty Computation. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 483–513. Springer, 2022.
- [GM18] Hannes Groß and Stefan Mangard. A unified masking approach. *J. Cryptogr. Eng.*, 8(2):109–124, 2018.



- [GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *TIS@CCS 2016 Vienna*, page 3. ACM, 2016.
- [GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In *CT-RSA 2017*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017.
- [GMO01] Karine Gandolfi, Christophe Moutrel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In *CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.
- [GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In *CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.
- [GSM<sup>+</sup>19] Hannes Groß, Ko Stoffelen, Lauren De Meyer, Martin Krenn, and Stefan Mangard. First-Order Masking with Only Two Random Bits. In *TIS@CCS 2019*, pages 10–23. ACM, 2019.
- [HS13] Michael Hutter and Jörn-Marc Schmidt. The Temperature Side Channel and Heating Fault Attacks. In *CARDIS 2013*, volume 8419 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2013.
- [IKL<sup>+</sup>13] Yuval Ishai, Eyal Kushilevitz, Xin Li, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and David Zuckerman. Robust Pseudorandom Generators. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 576–588. Springer, 2013.
- [ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
- [KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
- [KMMS22] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. Automated Generation of Masked Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):589–629, 2022.
- [Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
- [KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - Statistical Independence and Leakage Verification. In *ASIACRYPT 2020*, Lecture Notes in Computer Science. Springer, 2020.
- [KSM22] David Knichel, Pascal Sasdrich, and Amir Moradi. Generic Hardware Private Circuits - Towards Automated Generation of Composable Secure Gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1), 2022.

- [MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):256–292, 2019.
- [MMW18] Lauren De Meyer, Amir Moradi, and Felix Wegener. Spin Me Right Round Rotational Symmetry for FPGA-Specific AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):596–626, 2018.
- [NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schl affer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptol.*, 24(2):292–321, 2011.
- [PR13] Emmanuel Prouff and Matthieu Rivain. Masking against Side-Channel Attacks: A Formal Security Proof. In *EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2013.
- [RBN<sup>+</sup>15] Oscar Reparaz, Beg ul Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In *CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.
- [SAK] SAKURA. Side-channel Attack User Reference Architecture. <http://satoh.cs.uec.ac.jp/SAKURA/index.html>.
- [SBHM20] Pascal Sasdrich, Beg ul Bilgin, Michael Hutter, and Mark E. Marson. Low-Latency Hardware Masking with Application to AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):300–326, 2020.
- [SBM21a] Aein Rezaei Shahmirzadi, Dusan Bozilov, and Amir Moradi. New First-Order Secure AES Performance Records. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):304–327, 2021.
- [SBM21b] Aein Rezaei Shahmirzadi, Dusan Bozilov, and Amir Moradi. New First-Order Secure AES Performance Records. *IACR Cryptol. ePrint Arch.*, page 37, 2021.
- [SM15] Tobias Schneider and Amir Moradi. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In *CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.
- [SM21a] Aein Rezaei Shahmirzadi and Amir Moradi. Re-Consolidating First-Order Masking Schemes Nullifying Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):305–342, 2021.
- [SM21b] Aein Rezaei Shahmirzadi and Amir Moradi. Second-Order SCA Security with almost no Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):708–755, 2021.
- [Sug19] Takeshi Sugawara. 3-Share Threshold Implementation of AES S-box without Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):123–145, 2019.
- [Tri03] Elena Trichina. Combinational logic design for AES subbyte transformation on masked data. *IACR Cryptol. ePrint Arch.*, 2003:236, 2003.
- [WM18] Felix Wegener and Amir Moradi. A Note on Transitional Leakage When Masking AES with Only Two Bits of Randomness. *IACR Cryptol. ePrint Arch.*, 2018:1117, 2018.

## 5.3 Low-Latency Hardware Private Circuits

### Publication Data

David Knichel and Amir Moradi. Low-Latency Hardware Private Circuits. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1799–1812. ACM, 2022

*This work is reproduced here with permission. Due to an agreement between Ruhr University Bochum and ACM, this work is part of ACM Open. As a result, this work is licensed under a Creative Commons Attribution 4.0 International License and copyright is held by the authors. The author of this thesis is also an author of this research paper.*

**Content.** As of the time of publication, this work was the first to introduce AND gadgets that are instantiable at arbitrary security orders and trivially composable under the notion of PINI, while only introducing a single cycle latency into the design. With our construction – called HPC3 – we can significantly lower the latency requirements compared to the state of the art, making it highly favorable in the context of low-latency application like for example fast memory encryption. We further introduce HPC3<sup>+</sup> which guarantees secure composability under the simultaneous presence of glitches *and* transitions, even in edge cases, where there is a single gadget executed in a loop. Next to providing formal arguments for the security and composability of our construction, we present various case studies and extensive experimental leakage assessment.

**Contribution.** The author of this thesis is the principal author of this work. The author of this thesis likes to thank his co-author who is also his thesis supervisor.





# Low-Latency Hardware Private Circuits

David Knichel 

firstname.lastname@rub.de  
Ruhr University Bochum  
Horst Görz Institute for IT Security  
Bochum, Germany

Amir Moradi 

firstname.lastname@uni-koeln.de  
University of Cologne  
Institute for Computer Science  
Cologne, Germany

## ABSTRACT

Over the last years, the rise of the Internet of Things (IoT), and the connection of mobile – and hence physically accessible – devices, immensely enhanced the demand for fast and secure hardware implementations of cryptographic algorithms which offer thorough protection against Side-Channel Analysis (SCA) attacks. Among a variety of proposed countermeasures against SCA, *masking* has transpired to be a promising candidate, attracting significant attention in both, academia and industry. Here, abstract adversary models have been derived, aiming to accurately model real-world attack scenarios, while being sufficiently simple to enable formally proving the SCA resilience of masked implementations on an algorithmic level. In the context of hardware implementations, the *robust probing model* has become highly relevant for proving SCA resilience due to its capability to model physical defaults like glitches and data transitions. As constructing a correct and secure masked variant of large and complex circuits is a challenging task, a new line of research has recently emerged, aiming to design small, masked subcircuits – realizing for instance a simple AND gate – which still guarantee security when composed to a larger circuit. Although several designs realizing such composable subcircuits – commonly referred to as *gadgets* – have been proposed, negligible research was conducted in order to find trade-offs between different overhead metrics, like randomness requirement, latency, and area consumption.

In this work, we present HPC3, a hardware gadget which is trivially composable under the notion of PINI in the glitch-extended robust probing model. HPC3 realizes a two-input AND gate in one clock cycle which is generalized for any arbitrary security order. Existing state-of-the-art PINI-gadgets either require a latency of two clock cycles or are limited to first-order security. In short, HPC3 enables the designer to trade double the randomness for half the latency compared to existing gadgets, providing high flexibility and enabling the designer to gain significantly more speed in real-time applications.

## CCS CONCEPTS

• Security and privacy → Side-channel analysis and countermeasures.





This work is licensed under a Creative Commons Attribution-NonCommercial International 4.0 License.

CCS '22, November 7–11, 2022, Los Angeles, CA, USA  
© 2022 Copyright held by the owner/author(s).  
ACM ISBN 978-1-4503-9450-5/22/11.  
<https://doi.org/10.1145/3548606.3559362>

## KEYWORDS

SCA-resilient Hardware; Composable Gadgets; Hardware Private Circuits

### ACM Reference Format:

David Knichel  and Amir Moradi . 2022. Low-Latency Hardware Private Circuits. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*, November 7–11, 2022, Los Angeles, CA, USA. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3548606.3559362>

## 1 INTRODUCTION

Since the first seminal description of Side-Channel Analysis (SCA) in [32, 33], a significant increase in physically accessible devices has further driven demand for effective protection against physical attacks. At the same time, a wide variety of different exploitable side channels have been revealed, including timing [32], power consumption [33], electromagnetic (EM) emanations [21], or temperature and heat dissipation [26], posing a wide-ranging and divers threat to cryptographic implementations. Despite the increasing effort which has been committed to developing countermeasures against SCA attacks during the last decades, it remains a challenging and complex task to design secure hardware implementations offering a sufficient security level on the one hand, and high efficiency on the other.

Among a wide range of emerged SCA countermeasures, *masking* has proven to be a highly promising candidate due to its well-founded theoretical background based on secret sharing and its well-understood security requirements [17]. Although extensive research in this area has been conducted and several masking schemes were proposed over the last years [23, 25, 27, 38, 40, 48], many of these schemes have been proven to suffer from either design flaws or invalid assumptions [36].

As a consequence, researchers started focusing on the establishment of formal models in order to abstractly define an SCA attacker's capabilities and to realistically model circuit behavior [5, 20, 27, 39]. The advantage of these models is twofold. On the one hand, they enable security verification of masked implementations on an abstract level, aiming to detect security flaws in an early stage of the design process. Here, a variety of tools, working at different abstraction levels and offering various accuracy, have been proposed [2–4, 9, 10, 12, 14, 30]. On the other hand, this formalization enables the design of masked constructions that are provably secure with respect to the defined adversary model. The ISW *d*-probing model [27] – and its extension to modeling physical defaults in hardware implementations [20] – has proven to be well suited to find such secure constructions, due to its convenient level of abstraction and its existing reduction to the *Noisy Leakage Model* [19], which

is considered to be the closest to reality with respect to accurately modeling leakage behavior.

Although different models for formal security verification have been established in the context of SCA, finding provably secure masking schemes remains a hard task for complex circuits and high security orders. As a remedy, different composability notions have been proposed which define properties that aim to guarantee security even when circuits are composed. Following a divide-and-conquer approach, this reduces the task of finding large secure circuits to the task of finding small subcircuits which are in conformity with the composability notions. In the context of the  $d$ -probing model, Non-Interference (NI)/Strong Non-Interference (SNI) [3, 4] and Probe-Isolating Non-Interference (PINI) [15] have been proposed, where SNI was introduced since the restrictions defined by NI were not sufficient to guarantee composability. As the scope of SNI was originally limited to single-output gadgets, Cassiers et al. provided an extension to cover multiple-output gadgets, but at the same time proposed PINI which further reduces the overhead necessary to construct composable gadgets [15]. In their recent work [16], Cassiers and Standaert further extended the adversary model to accurately cope with data transitions between different clock cycles and formally defined the notion of Output Probe-Isolating Non-Interference (O-PINI), which aims to achieve trivial composability under (i) transitions and (ii) transitions + glitches. In the course of the introduction of these wide variety of notions, several concrete gadgets have been presented, either realizing atomic logic functions, like AND or refresh gates [4, 9, 14–16, 24, 27, 35], or even arbitrary logic functions [31].

As embedded real-time applications become increasingly relevant – especially in the context of the rapidly growing IoT – and due to the high accessibility of the involved devices, designing fast and protected cryptographic hardware implementations is the key to guarantee a high level of security and sufficiently fast data processing. As a consequence, reducing the latency, introduced by masking a cryptographic hardware implementation, is an important task whose difficulty is mainly rooted in omitting leakage through physical defaults like *glitches* and *transitions* [16, 20]. As composable hardware gadgets for arbitrary security orders are restricted to atomic logic functions, reducing the latency of a gadget would have a significant effect on the latency of the over-all composed circuit. It is hence highly beneficial if we can further reduce the latency of existing schemes.

*Contributions.* In this work, we present HPC3, a low-latency hardware gadget, that offers  $d$ -th order security and is trivially composable under the notion of PINI in the glitch-extended robust probing model. Similar to HPC1 and HPC2 [14], it realizes a masked AND gate that can be instantiated for arbitrary security orders  $d$ . In contrast to HPC2, our newly introduced HPC3 only needs a single register stage instead of two, regardless of the security order, while doubling the randomness requirements. It hence enables the designer to trade double the randomness for half the latency, while sustaining the same security level and providing significantly more flexibility with respect to different use cases – for example in the context of fast memory encryption. To the best of our knowledge, HPC3 is the first glitch-robust composable gadget which is settled in the PINI framework and can be instantiated utilizing only

one register stage for any security order. We further show that, compared to HPC2, our construction itself leads to less chip area overhead – at the cost of a higher demand for fresh randomness.

Furthermore, we present HPC3<sup>+</sup>, a gadget that is directly based on HPC3 and offers trivial composability in the simultaneous presence of both transition and glitches. Eventually, we prove and formally verify our constructions, compare our work to state-of-the-art hardware gadgets, and explore multiple case studies and conduct experimental leakage assessments.

*Outline.* We start by elaborating all necessary theoretical concepts in Section 2. This includes a summary of all notations used throughout this work, a definition of the circuit and adversary model, and recapitulating different security and composability notions. In Section 3, we present our new gadget HPC3 and prove its conformity to the PINI notion in the glitch-extended robust probing model. Here, we further present HPC3<sup>+</sup> and prove its trivial composability under transitions paired with glitches, before we compare our designs to state-of-the-art trivially-composable hardware gadgets in Section 4. After we conduct analyses on an extensive list of case studies, i.e., different cryptographic implementations, and perform experimental leakage assessments in Section 5, we conclude our work in Section 6.

## 2 BACKGROUND

### 2.1 Notations

We denote random variables by capital letters, e.g.,  $X \in \mathbb{F}_2$  denotes a binary random variable. Further, we use bold letters like  $\mathbf{X}$  to denote sets. Initializations of random variables are denoted by small letters, while the probability that  $X$  takes  $x$  is written as  $\Pr[X = x]$ . Moreover, we indicate the  $i$ -th input to a function with subscript  $i$  while superscripts identify shares of random variables. Hence, the  $s$ -th share of the  $i$  input to a function is denoted as  $X_i^s$ . Let further  $|\mathbf{X}|_i$  denote the number of shares in a set  $\mathbf{X}$  corresponding to  $X_i$ . When masking a function  $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$  with  $t$  shares per input, the set containing all input shares is given as  $Sh(\mathbf{X}) = [X_0^0, X_0^1, \dots, X_0^{t-1}, X_1^0, \dots, X_{n-1}^{t-1}]$ . In the same manner, for a set of share indices  $\mathbf{I} \subseteq [0, \dots, t-1]$ ,  $Sh(\mathbf{X})^{\mathbf{I}}$  denotes the set of all input shares  $X_i^s$ , with  $0 \leq i < n$  and  $s \in \mathbf{I}$ . Eventually,  $P_W$  denotes the (extended) probe on a wire  $W$ , while drawing a value  $X$  uniformly and at random from a set  $S$  is denoted as  $X \xleftarrow{\$} S$ .

### 2.2 Circuit Model

As originally considered in [27] and later extended in [20], any stateful and deterministic circuit  $C$  is modeled as a Directed Acyclic Graph (DAG)  $G_C = \{\mathcal{V}, \mathcal{E}\}$  with  $\mathcal{V}$  being the set of vertices and  $\mathcal{E}$  the set of edges in  $G_C$ . The edges represent wires carrying elements of  $\mathbb{F}_2$  while the vertices are combinational gates such as AND and XOR, or memory gates, i.e., registers. Memory gates will output the previous input to the gate on any circuit invocation while storing the input for the next invocation. Eventually,  $G_C$  realizes a Boolean function  $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ .

As this model lacks the notion of cycling connections in a circuit, and the ability to model different executions of the same physical gate, it has been extended towards a more specific circuit model in [16]. This model introduces so-called *structural gates* which

among its functionality, public and secret parameters, also captures the latency of a gate. *Structural wires* are then defined as wires connecting structural gates, and finally a *structural circuit* is defined as a directed graph whose nodes are structural gates and whose edges are structural wires.

The definition of a structural circuit can then be extended to cover states in different clock cycles by defining a state of each wire and gate for every clock cycle:

**DEFINITION 2.1 (CIRCUIT EXECUTION).** *A Circuit Execution of a structural circuit  $C = (G, W)$  for the set of cycles  $T$  is a directed graph  $G_C = (\mathcal{V}, \mathcal{E})$  with  $\mathcal{V} \in G \times T$  and  $\mathcal{E} \in W \times T$  where wires connect the gates according to their latency. Here,  $G$  and  $W$  denote the structural gates and wires of  $C$ .*

Note that in this model, *combinational gates* – like AND and XOR – are structural gates with latency  $l = 0$ , realizing the corresponding Boolean function, while registers are structural gates realizing the identity function and having latency  $l = 1$ . Nevertheless, structural gates are defined in a more general way, possibly containing several register stages.

Thanks to the previous research conducted in [16] and due to the share isolating property of PINI, we can conveniently limit all our considerations to the simpler original circuit model while achieving secure composition under transitions in the more complex model of circuit executions.

*Encoded Circuit Model.* In order to restrict the adversarial probing solely to the masked circuit, and to exclude the masking and unmasking of the input data, the process of secure computation is divided into three steps. As defined in [1], a *circuit compiler* is defined by a tuple of three algorithms (COMPILE, ENCODE, DECODE), which are defined as follows.

- The COMPILE algorithm is deterministic and takes as input a structural circuit  $C$  and outputs a randomized (masked) circuit  $\tilde{C}$ .
- ENCODE is a probabilistic algorithm that takes as input  $X$  and outputs the encoded input  $\tilde{X}$ . This encoded input corresponds to the shared representation of the data, i.e.,  $\tilde{X} = Sh(X)$ , which – in our case – is derived by means of Boolean masking.
- Eventually, DECODE is a deterministic algorithm that takes encoded data  $\tilde{Y}$  and decodes/unshares it to achieve  $Y$ .

As a result, these algorithms enable sharing of the initial input data (through ENCODE), a computation on the shared representation of the input (through  $\tilde{C}$ ) and an unsharing of the result (through DECODE), such that  $Y \leftarrow DECODE \circ \tilde{C} \circ ENCODE(X)$  for  $Y \leftarrow C(X)$ , while the adversary is restricted to only make observations within  $\tilde{C}$ . It means that the algorithms ENCODE and DECODE are known to the adversary, but neither the output of ENCODE nor the input of DECODE.

### 2.3 Boolean Masking

Following the approach of secret sharing, a *Boolean masking* of a secret  $X \in \mathbb{F}_2^n$  is a set  $X \in \mathbb{F}_2^{n \times s}$  of  $s$  independent secret shares

$X^i \in \mathbb{F}_2^n$ ,  $0 \leq i < s$ , such that  $X = \bigoplus_{i=0}^{s-1} X^i$ . This is commonly

derived by independently drawing  $X^i \xleftarrow{\$} \mathbb{F}_2^n$ , for  $0 \leq i < s - 1$  and

calculating the remaining share as the XOR sum of all other shares:

$$X^{s-1} = \bigoplus_{i=0}^{s-2} X^i.$$

Following the definition described in Section 2.2, this step describes the ENCODE algorithm in the case of Boolean masking. A direct implication of Boolean masking necessitates splitting sensitive data in at least  $d + 1$  shares in order to achieve  $d$ -th order security.

### 2.4 Adversary Model

Following the definition in Section 2.2, for the remainder of this work, we assume that an adversary's access to computations is limited to  $\tilde{C}$  and that the execution of ENCODE and DECODE remain unavailable.

*d-Probing Model.* The standard  $d$ -probing model, introduced by Ishai et al. in [27], grants an adversary the ability to probe up to  $d$  wires of a circuit. Here, every logic gate acts as a synchronization element, so every wire is considered stable, carrying only the result of the driving gate under the current assignment of the primary inputs. Intuitively, security in this model is given, if an adversary is not able to receive any information about sensitive (unshared) data by observing the joint distribution over maximal  $d$  probed wires.

*Glitch-Extended Probing.* Since the standard probing model does not cover the modeling of physical defaults occurring in hardware implementations, it was extended by Faust et al. in [20]. In their work, the authors introduced the *robust probing model* which aims to extend the probes enabling them to capture leakage originating from (i) data transitions at registers, (ii) coupling effects, i.e. dependencies between adjacent wires, and (iii) glitches.

The concept of *glitches* describes signal recombination caused by different delay paths in a digital circuit. Hence, in practice, a single probe on a wire is not only able to observe the intended and stable output signal of its driving gate, but possibly a recombination of several upstream signals, up to the last synchronization point, i.e., registers output or primary inputs. To capture the worst case in this context, the glitch extension of a probe is hence the set of probes placed on all these synchronization points.

This is formally defined through Algorithm 1. The glitch extension of a probe is defined recursively and will either return the union over all glitch-extended probes placed on the input of the driving combinational gate, or return the probe itself in case the probe is placed on an output of a register or a primary input. Eventually, the extension of a set of standard probes is simply derived by uniting all corresponding glitch-extended probes.

*Transition-Extended Probing.* Considering the definition of a circuit execution, given in Definition 2.1, the transition-extended probes on a *structural gate* – which is formally defined in [16] by the fact that all its executions are identical except for a shift in time, i.e., in clock cycles – are all non-extended probes  $P_W$  and additionally the sets  $\{(W, t - 1), (W, t)\}$ . This means the joint distribution of probes in two consecutive clock cycles while  $\{(W, t - 1), (W, t)\}$  belongs to the same execution of the probed structural gate. Here,  $t$  refers to a clock cycle and  $(W, t)$  refers to the state of  $W$  during this specific clock cycle, i.e., a node in the circuit execution given in Definition 2.1. For example, a non-extended probe on a register output would extend to two probes: on its input and output wires.

**Algorithm 1:** Glitch Extension

---

**Input** : Non-extended probe  $P \in \mathbb{F}_2$   
**Output** : Glitch-extended probes  $P^E \in \mathbb{F}_2^p, p > 0$

```

1 if  $P$  is placed on an output of a combinational gate then
2    $P^E \leftarrow \bigcup_{0 \leq i < n} \text{glitch-extend}(P_i)$  // where  $P_i, 0 \leq i < n$ 
   // are all inputs to the driving gate
3 else
4   if  $P$  is placed on an output of a register or on a primary input
5     then
6     |  $P^E \leftarrow \{P\}$ 
7   end

```

---

*Transition+Glitch-Extended Probing.* Combining transition- and glitch-based probe extensions is straightforward. First, all non-extended probes are glitch-extended, before every resulting probe becomes transition-extended. This corresponds to the physical property that a propagation of a glitch may depend on both the new and the previous value of the wire.

*d-Probing Security.* In the (robust) probing model, an adversary is granted the ability to probe up to  $d$  wires of a masked circuit  $\tilde{C}$ . Naturally,  $d$ -probing security is achieved iff an adversary with these capabilities, i.e., an adversary in the  $d$ -probing model, does not learn anything about the processed secrets [27]:

**DEFINITION 2.2** (*d-PROBING SECURITY*). *A masked circuit  $\tilde{C}$  – derived by compiling  $C$  with secret input  $X$  – achieves  $d$ -probing security iff for any set of probes  $P, |P| \leq d$ , the joint distribution over all observations  $Q$  made by (extended) probes  $P$  is statistically independent of any secret  $X$ , i.e., the following holds for all possible assignments to variables in  $Q$  and  $X$ :*

$$\Pr[Q|X] = \Pr[Q]$$

## 2.5 Circuit Composition

Several security and composability notions have been proposed in recent years, aiming to enable efficient masking of large and complex circuits. Before we give an overview of these notions, we start by introducing the concepts of *probe simulation* and *probe propagation*, paving the ground for the subsequent definitions.

*Probe Simulatability.* The concept of *probe simulatability* [15, 27] helps to argue about dependencies between probes and input shares to a masked/encoded circuit. Its definition is given in the following.

**DEFINITION 2.3** (*PERFECT PROBE SIMULATION*). *Given a set of (extended) probes  $P \in \mathbb{F}_2^t$  on an encoded circuit  $\tilde{C}$ ,  $P$  is said to be perfectly simulatable by a set  $S$  of input shares iff there exist a simulator  $SIM$ , such that for any values of the inputs to  $\tilde{C}$ , the joint probability distribution over  $P$  and  $SIM(S)$  are equal, where  $SIM(S) : \mathbb{F}_2^{|S|} \mapsto \mathbb{F}_2^t$  with input  $S \subseteq Sh(X)$  is a probabilistic polynomial time (p.p.t.) simulator.*

*Probe Propagation.* The concept of *Probe Propagation* – initially introduced by Cassiers et al. in [15] – is closely related to simulatability, and defines which wires are necessary to perfectly simulate a probe placed on  $\tilde{C}$ . Intuitively, it describes, how leakage is traversed

backwards throughout a circuit, beginning from the point where the probe is placed. Restricting probe propagation has proven to be a key factor to guarantee composability.

Considering a probe  $P \in \mathbb{F}_2$  on a subcircuit,  $P$  is said to propagate into an input of the subcircuit, if the input is required to perfectly simulate  $P$ . When considering composability notions, propagation of internal and output probes of a gadget are restricted following a well defined set of rules and allowing to make statements of the overall circuit. For this, the propagated probes can be derived by iteratively substituting a probe by its propagated variant on the gadgets’ inputs until the overall circuit’s input is reached.

*Non-Interference (NI).* As directly designing masked circuits which achieve  $d$ -probing security has proven to be hard for large and complex functions, different composability notions have been introduced as a remedy. These notions aim to define sufficient properties which a masked circuit must fulfill in order to provide  $d$ -th order security in the probing model when composed to a larger circuit. Following a divide-and-conquer approach and utilizing atomic logic functions – typically AND and XOR – large circuits can be derived by composing masked versions of these atomic gates – commonly referred to as *gadgets*.

Utilizing the concept introduced above, and as initially elaborated in [3], NI aims to restrict probe propagation to a certain set of input shares:

**DEFINITION 2.4** (*d-NON-INTERFERENCE (NI)*). *A masked circuit  $\tilde{C}$  provides  $d$ -Non-Interference iff for any probe set  $|P|$  of  $t \leq d$  probes, there exists a set  $S$  of input shares with  $|S|_{v_i} \leq t$  such that  $P$  can be perfectly simulated by  $S$ .*

*Strong Non-Interference (SNI).* The notion of SNI [4] has been introduced in order to correct composability flaws regarding NI and restricts probe propagation even further:

**DEFINITION 2.5** (*d-STRONG NON-INTERFERENCE (SNI)*). *A masked circuit  $\tilde{C}$  provides  $d$ -Strong Non-Interference iff for any probe set  $P$ , containing  $t = t_1 + t_2 \leq d$  probes, where  $t_1$  probes are placed on internal wires and  $t_2$  on output wires, there exists a simulation set  $S$  of input shares with  $|S|_{v_i} \leq t_1$ , such that  $P$  can be perfectly simulated by  $S$ .*

Here, output probes are not allowed to propagate into any input shares, hence resolving the flaws with respect to composability discovered for NI [4], but implication of secure composition was originally still restricted to single-output gadgets.

*Probe-Isolating Non-Interference (PINI).* In [15], it has been shown that the original definition of SNI can be extended to cover multiple-output gadgets as well. Nonetheless, in the same work, the authors introduced the notion of PINI, which elegantly isolates probe propagation within single share domains, enabling trivial implementations of linear functions and reducing entropy and area requirements compared to SNI, while guaranteeing straightforward composability.

Similar to Domain-Oriented Masking (DOM), in the context of PINI, a specific share domain is assigned to every input and output share. Now, in order to be PINI, every output probe is only allowed to propagate within its own share domain (share index)



while propagation of every internal probe is limited to a single (but arbitrary) share domain:

**DEFINITION 2.6** (*d*-PROBE-ISOLATING NON-INTERFERENCE (PINI)). *Let  $\mathbf{P}_1$  be the set of internal probes with  $|\mathbf{P}_1| = t_1$ . Let further  $\mathbf{I}_O$  be the index set assigned to the output wires probed by  $\mathbf{P}_O$  with  $|\mathbf{I}_O| = t_2$ .*

*A masked circuit  $\tilde{C}$  provides *d*-Probe-Isolating Non-Interference iff for every  $\mathbf{P} = \mathbf{P}_1 \cup \mathbf{P}_O$  with  $t_1 + t_2 \leq d$ , there exists a set  $\mathbf{I}_1$  of circuit indices with  $|\mathbf{I}_1| \leq t_1$  such that  $\mathbf{P}$  can be perfectly simulated by  $S = Sh(X)^{\mathbf{I}_1 \cup \mathbf{I}_O}$ .*

Conveniently, this definition directly implies that the trivial implementation of linear functions with  $d+1$  shares, i.e., the share-wise application of the unshared function, is *d*-PINI even in the glitch-extended robust probing model. Consequently, linear functions do not introduce any additional latency or entropy overhead into the design.

PINI is invariant under composition, i.e., a circuit composed of gadgets fulfilling *d*-PINI is itself *d*-PINI. This is true, if gadgets are carefully connected, i.e., if an output with a certain share index is only connected to an input with the same share index. Therefore, since *d*-PINI implies *d*-probing security, the resulting circuit will be secure in the *d*-probing security model and will reveal no information about any secret value under *d* (extended) probes.

In order to achieve trivial composition under transitions, an extension to the original definition of PINI, i.e., O-PINI, was recently introduced in [16]. It deals with adjacent executions of the same gadget, i.e., if the input of a gadget depends on the output of itself. As for simulating the second execution of the gadget, the simulator may need to also simulate outputs of the first gadget execution, this leads to an effective probe extension of internal probes to additional probes on the output. As a consequence, the probed wires may propagate into more circuit shares than allowed, causing a reduction in the security level of the design. Naturally, a possible solution would be to avoid adjacent execution of gadgets, for example by means of an ‘empty/dummy’ clock cycle between executions. As this would introduce composability restrictions, which make additional design verifications necessary, O-PINI was introduced in order to guarantee trivial composability. O-PINI copes with the issue of adjacent executions by additionally enforcing simulatability of every output wire whose share index is also a share index in the simulation set:

**DEFINITION 2.7** (*d*-OUTPUT PROBE-ISOLATING NON-INTERFERENCE (O-PINI)). *Let  $\mathbf{P}_1$  be the set of internal probes with  $|\mathbf{P}_1| = t_1$ . Let further  $\mathbf{I}_O$  be the index set assigned to the output wires probed by  $\mathbf{P}_O$  with  $|\mathbf{I}_O| = t_2$ .*

*A (unrolled) masked circuit  $\tilde{C}$  provides *d*-Probe-Isolating Non-Interference iff for every  $\mathbf{P}$  with  $t_1 + t_2 \leq d$ , there exists a set  $\mathbf{I}_1$  of circuit indices with  $|\mathbf{I}_1| \leq t_1$  such that  $\mathbf{P} = \mathbf{P}_1 \cup \mathbf{P}_O \cup \mathbf{O}^{\mathbf{I}_1}$  can be perfectly simulated by  $S = Sh(X)^{\mathbf{I}_1 \cup \mathbf{I}_O}$ .*

Note that, for the sake of simplicity and in contrast to [16], we restrict the definition of *d*-O-PINI to unrolled designs (which in [16] is called *pipelined*), i.e., designs where no feedback loops exist. Due to their implication of trivial composability, all our constructions in this work are based on the PINI and O-PINI notions.

## 2.6 Automated Masking

Gadgets fulfilling composability notions like PINI are well suited for automated generation of masked hardware. By simply synthesizing the implementation based on a library that only includes logic gates for which a corresponding gadgets exist, the implementation can be masked by substituting every gate in the resulting netlist with those gadgets. Since composability notions elegantly guarantee (robust) probing security when gadgets are composed, the overall design will be provably secure.

The recently introduced software tool AGEMA [29] realizes such a transformation by first representing the netlist as a graph, before translating it into a Mealy machine consisting of a combinational circuit and a single main register stage. The designer can then select which parts of the netlist should be masked and specify the gadget family to apply. In the *naive* approach, every gate in the netlist is then simply substituted by the corresponding gadget as explained. As these gadgets introduce additional latency into the design, the correct functionality of the circuit is ensured by automatically applying pipelining or clock gating. We utilized AGEMA in this work in order to construct our case studies and to offer a fair comparison to state-of-the-art composable gadgets.

## 3 LOW-LATENCY HARDWARE PRIVATE CIRCUITS

In this section, we present the algorithm describing our low-latency Hardware Private Circuits (HPC3), formally prove its security and composability and give exemplary schematics for the first and second security orders, before we extend it to be composable under transitions and glitches.

### 3.1 Glitch-Robust Variant

HPC3 is described in Algorithm 2. Here, every cross-domain term  $X^i \cdot Y^j$  is blinded by the sum of two freshly drawn random masks  $R'_{ij} \oplus R''_{ij}$ , where  $R'_{ij} = R'_{ji}$  and  $R''_{ij} = R''_{ji}$ , resulting in a randomness requirement of  $2 \cdot (d \cdot (d + 1)) / 2 = d \cdot (d + 1)$  and a latency of a single clock cycle to achieve *d*-th order security in the glitch-extended probing model. The composability under the notion of PINI – utilizing only a single register stage – is achieved through blinding  $Y^j$  by  $R'_{ij}$  before multiplying  $X^i$ , and at the same time blinding the corresponding correction term  $\overline{X^i} \cdot R'_{ij}$  by another fresh randomness  $R''_{ij}$ . This way, a single extended output probe is never able to reveal  $X^i$  and  $Y^j$ .

Intuitively, this means that probes on different cross-domain terms can always be simulated as they are independently blinded and an adversary would always need two probes on  $X^i \cdot Y^j$  and  $X^j \cdot Y^i$  to reveal information about domain *i* and *j*. Yet, these two probes would always be placed on share domain *i* and share domain *j*, which is in conformity with the PINI security notion. We formalize this argument by proving Theorem 3.1 as follows.

**THEOREM 3.1.** *HPC3– with security parameter *d* – provides a correct and PINI-secure circuit in the glitch-extended *d*-probing model.*

**Algorithm 2:** HPC3 Multiplication

---

**Input** :  $Sh(X) = [X^0, \dots, X^d]$ ,  $Sh(Y) = [Y^0, \dots, Y^d] \in \mathbb{F}_2^{d+1}$

**Output**:  $Sh(Z) = [Z^0, \dots, Z^d] \in \mathbb{F}_2^{d+1}$

/\* valid sharings of  $X, Y, Z = X \cdot Y \in \mathbb{F}_2$  \*/

```

1 for  $i = 0$  to  $d - 1$  do
2   for  $j = i + 1$  to  $d$  do
3      $R'_{ij} \xleftarrow{\$} \mathbb{F}_2$ ,  $R''_{ij} \xleftarrow{\$} \mathbb{F}_2$ 
4      $R'_{ji} \leftarrow R'_{ij}$ 
5      $R''_{ji} \leftarrow R''_{ij}$ 
6   end
7 end
8 for  $i = 0$  to  $d$  do
9   for  $j = 0$  to  $d$  do
10    if  $i \neq j$  then
11       $U'_{ij} \leftarrow \text{Reg}[Y^j \oplus R'_{ij}]$ 
12       $U''_{ij} \leftarrow \text{Reg}[\overline{X^i} \cdot R'_{ij} \oplus R''_{ij}]$ 
13       $C_{ij} \leftarrow \text{Reg}[X^i] \cdot U'_{ij} \oplus U''_{ij}$ 
14    end
15  end
16 end
17 for  $i = 0$  to  $d$  do
18    $Z^i \leftarrow \text{Reg}[X^i \cdot Y^i] \oplus \bigoplus_{0 \leq j \leq d} (C_{ij})$ 
19 end

```

---

**PROOF.**

*Correctness.* First, we prove the correctness of the construction. For this, we discard all registers in Algorithm 2. Since

$$\begin{aligned}
C_{ij} &= X^i \cdot (Y^j \oplus R'_{ij}) \oplus \overline{X^i} \cdot R'_{ij} \oplus R''_{ij} \\
&= X^i \cdot Y^j \oplus X^i \cdot R'_{ij} \oplus \overline{X^i} \cdot R'_{ij} \oplus R''_{ij} \\
&= X^i \cdot Y^j \oplus R'_{ij} \oplus R''_{ij}
\end{aligned}$$

and  $R'_{ij} = R'_{ji}$ ,  $R''_{ij} = R''_{ji}$ , it holds that

$$\begin{aligned}
Z &= \bigoplus_{0 \leq i \leq d} Z^i \\
&= \bigoplus_{0 \leq i \leq d} \left( X^i \cdot Y^i \oplus \bigoplus_{0 \leq j \leq d, j \neq i} (X^i \cdot Y^j \oplus R'_{ij} \oplus R''_{ij}) \right) \\
&= \bigoplus_{0 \leq i \leq d} \left( \bigoplus_{0 \leq j \leq d} X^i \cdot Y^j \right) \oplus \bigoplus_{0 \leq i \leq d} \left( \bigoplus_{0 \leq j \leq d, i \neq j} R'_{ij} \oplus R''_{ij} \right) \\
&= \bigoplus_{0 \leq i \leq d} \left( \bigoplus_{0 \leq j \leq d} X^i \cdot Y^j \right) \\
&= \left( \bigoplus_{0 \leq i \leq d} X^i \right) \cdot \left( \bigoplus_{0 \leq j \leq d} Y^j \right) = X \cdot Y.
\end{aligned}$$

*PINI.* Next, we prove PINI security in the glitch-extended  $d$ -probing model by considering every relevant case of probe placement and arguing about simulatability.

- I. A glitch-extended probe on the input to  $U'_{ij}$ , i.e.,  $P_{U'_{ij}} = [Y^j, R'_{ij}]$ , can be perfectly simulated by  $Y^j$  and tossing a fair coin  $R'_{ij} \xleftarrow{\$} \mathbb{F}_2$ .
- II. The glitch-extended probes  $P_{U''_{ij}} = [X^i, R'_{ij}, R''_{ij}]$  can be simulated by  $X^i$  and drawing two random bits  $R'_{ij}, R''_{ij} \xleftarrow{\$} \mathbb{F}_2$ . If additionally  $P_{U''_{ji}}$  needs to be simulated, this can straightforwardly be done by adding  $X^j$  to the simulation set.
- III. A glitch-extended probe on  $C_{ij}$ , i.e.,

$$P_{C_{ij}} = [X^i, Y^j \oplus R'_{ij}, \overline{X^i} \cdot R'_{ij} \oplus R''_{ij}]$$

can be simulated by  $[X^i, R'_{ij}, R''_{ij}]$ , where  $R'_{ij}, R''_{ij} \xleftarrow{\$} \mathbb{F}_2$ . If additionally  $P_{C_{ji}}$  needs to be simulated – which lies in share domain  $j$  – this can be done by adding  $X^j, Y^j$  and  $Y^i$  to the input set of the simulator. As now two probes are considered, this is in conformity to the PINI notion. All other probes can be handled in the same manner but completely independent, as every share domain is blinded with completely fresh (and hence independent) randomness.

- IV. A glitch-extended output probe

$$P_{Z^i} = [X^i, Y^i] \cup \left\{ \bigcup_{0 \leq j \leq d, i \neq j} C_{ij} \right\}$$

can be simulated by  $X^i$  and  $Y^i$  and drawing  $d$  times  $R'_{ij} \xleftarrow{\$} \mathbb{F}_2$  and  $R''_{ij} \xleftarrow{\$} \mathbb{F}_2$  (to simulate  $C_{ij}$ ). As by construction, two outputs have at most one cross domain in common, and following the same argument as given above in step III, adding another probe would result in requiring (next to some additional random bits) inputs from only one other domain. This is in conformity to the PINI notion.  $\square$

In Figure 1, we provide a schematic overview of our construction, configured for the first security order and requiring two freshly drawn random bits. In order to highlight at which points the same randomness is introduced, we simply denote  $R'_{01} = R'_{10}$  and  $R''_{01} = R''_{10}$  by  $R'$  and  $R''$ , respectively. Furthermore, we give the schematics for the first circuit share of our second-order ( $d = 2$ ) design in Figure 2.

In addition to proving PINI security in the glitch-extended probing model for general security order  $d$ , we further formally verified our construction up to the fourth order utilizing SILVER [30] – an open-source Binary Decision Diagram (BDD)-based software tool for formally verifying different security and composability notions under the glitch-extended probing model. We choose SILVER over other existing verification tools as, at the time we conducted our research, it was the only tool available for verifying the notion of PINI. In parallel to the publication of this work, IronMask [8] became available as an alternative tool.

### 3.2 Iterated Glitch+Transition-Robust Variant

As elaborated in [16], fulfilling the notion of PINI may not be sufficient to guarantee security under transitions in an iterative design, i.e., in a design where the same gadget instantiation is executed

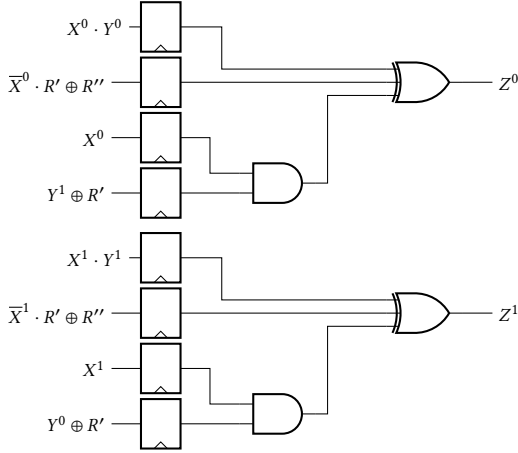


Figure 1: Schematics for HPC3 instantiated for  $d = 1$ .

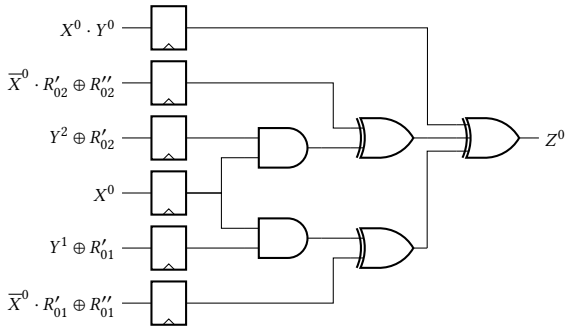


Figure 2: Schematics for HPC3 instantiated for  $d = 2$ , only the part of the circuit required to generate an output share  $Z^0$ .

several times and the input of one iteration depends on the output of a preceding one. The reason is depicted in Figure 3. During an execution of gadget  $G$ , an adversary places an internal, transition-extended probe  $P_1$  on the gadget, which w.l.o.g. propagates into share domain 1. As depicted in Figure 3, in order to simulate the probe  $P_1$ , the output  $Z^1$  of the prior execution is needed, which is equivalent to placing a second probe  $P_2$  onto  $Z^1$ . Now PINI only guarantees that the new probe set  $\mathbf{P} = \{P_1, P_2\}$  can be simulated utilizing two input shares instead of one (while only one probe is counted), possibly decreasing the design’s security order. As a remedy, the authors introduced the notion O-PINI which, similar to the methodology described in [16], enables us to extend HPC3 to also be trivially composable under (iterated) transitions and glitches. The algorithm of our iterated transition+glitches-robust variant is given as HPC3<sup>+</sup> in Algorithm 3. Here, each output is blinded by an individual fresh mask  $M_i$ ,  $0 \leq i \leq d$  before saved in an output register, intuitively enabling simpler simulation of the output wires. For the sake of visualization, Figure 4 depicts the schematic of a part of the circuit (for an output share) of HPC3<sup>+</sup> configured for  $d = 2$ .

Due to the share-separating nature of O-PINI, and based on Lemma 1 and Lemma 2 given in [16], we can leverage the fact

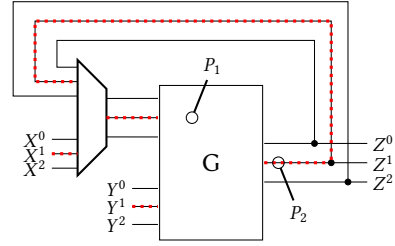


Figure 3: Schematic probe propagation between different iterations of the same gadget instantiation

that our glitch+transition-robust HPC3<sup>+</sup> gadget is unrolled, which means that there are no loops within the gadget, enabling us to prove *iterated glitch+transition-robust* composability, i.e., composability under transitions+glitches and subsequent executions of the same hardware gadget, by conveniently staying in our simplified circuit model and proving perfect simulatability in conformity with O-PINI, as given in Definition 2.7.

We now prove the security and trivial composability of our HPC3<sup>+</sup> gadget in the presence of both, iterated transitions *and* glitches:

**THEOREM 3.2.** *HPC3<sup>+</sup> is iterated glitch+transition-robust t-O-PINI for  $t \leq d$ , where  $d$  is the security order.*

**PROOF.**

**Correctness.** The correctness of the construction is directly implied by the correctness of HPC3 and the fact that  $\bigoplus_{0 \leq i \leq d} M_i = 0$ .

O-PINI. The proof follows the same argument as the proof given in [16]. Based on this and the fact that our design is unrolled, we only have to prove that one execution of HPC3<sup>+</sup> is glitch-robust O-PINI and we can stay in our simple circuit model. Without loss of generality, we restrict our analysis to the most powerful probes and use the probe simulator for HPC3 considered in the proof of Theorem 3.1. Intuitively, the additional refresh after executing HPC3 enables a simulator to simulate additional output probes possibly caused by subsequent executions of the gadget.

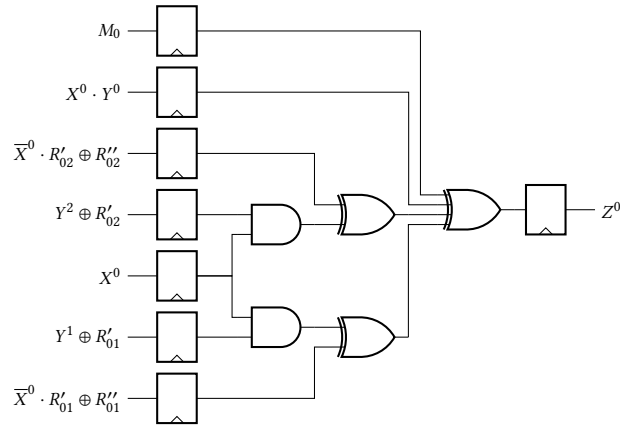
- I. Glitch-extended internal probes on  $W_i$  are simulated correctly by the simulator for HPC3.
- II. Glitch-extended probes on  $P_{W_i \oplus M_i} = [W_i, M_i]$  can be simulated by running the simulator of HPC3<sup>+</sup> to generate  $W_i$  – which can be simulated using only shares from share domain  $i$  – and by drawing a fresh random bit  $M_i \xleftarrow{\$} \mathbb{F}_2$ .
- III. For glitch-extended probes  $P_{Z_i} = [W_i \oplus M_i]$  which are added to the output due to the propagation of an internal probe, i.e.,  $i \in \mathbf{O}^H$  given in Definition 2.7 of Section 2.5, we consider the following cases:
  - i. If not all intermediate values of  $W_i$  have been simulated, the simulator simply flips a fair coin and outputs the result.
  - ii. If all intermediate values of  $W_i$  have already been simulated (for example if there is another probe on  $W_i$ ), use these values and perform as specified by the algorithm.

**Algorithm 3:** HPC3<sup>+</sup> Multiplication

```

Input :  $Sh(X) = [X^0, \dots, X^d], Sh(Y) = [Y^0, \dots, Y^d] \in \mathbb{F}_2^{d+1}$ 
Output:  $Sh(Z) = [Z^0, \dots, Z^d] \in \mathbb{F}_2^{d+1}$ 
/* valid sharings of  $X, Y, Z = X \cdot Y \in \mathbb{F}_2$  */

1  $W \leftarrow \text{HPC3}(X, Y)$ 
2 for  $i = 0$  to  $d - 1$  do
3    $M_i \xleftarrow{\$} \mathbb{F}_2$ 
4 end
5  $M_d \leftarrow \bigoplus_{0 \leq i < d} M_i$ 
6 for  $i = 0$  to  $d$  do
7    $Z_i \leftarrow \text{Reg}[W_i \oplus \text{Reg}[M_i]]$ 
8 end
    
```



**Figure 4:** Exemplary circuit share of HPC3<sup>+</sup> instantiated for  $d = 2$

The simulation of output probes is correct, as for the worst case where there is a glitch-extended probe on  $M_d$  (and hence all  $M_i, 0 \leq i \leq d$  are observed enabling to unblind all  $P_{C_i}$ ), there are at most  $d - 1$  other probes. If one of these probes is  $P_{W_i}$  or  $P_{W_i \oplus S_i}$ , we can trivially simulate it with shares from share domain  $i$  and using the simulator of HPC3. If the probes are internal probes on HPC3, no glitch-extended probe on a register stage other than  $P_{W_i}$  or  $P_{W_i \oplus S_i}$  can observe more than one  $R'_{ij} \oplus R''_{ij}$  at once. As a consequence, maximal  $d - 1$  of the  $d$  terms  $R'_{ij} \oplus R''_{ij}$  are observed per  $i$ , and  $Z_i$  is blinded by one fresh random bit. The simulator can hence always simulate it by tossing a fair coin.  $\square$

**4 COMPARISON TO STATE-OF-THE-ART HPCs AND LOW-LATENCY DESIGNS**

A wide variety of Hardware Private Circuits (HPCs), i.e., trivially composable hardware gadgets, have been recently introduced, differing with respect to their supported security order, their robustness against glitches and transitions, their randomness requirements and their latency. In the following, we introduce all related designs, before we compare them with our low-latency variants. To ensure comparability with respect to composability guarantees,

we restrict our comparison to gadgets that are trivially composable in (i) the presence of glitches and/or (ii) transitions.

**4.1 Glitch-Robust Hardware Private Circuits**

HPC1 [14]. The HPC1 gadget – introduced by Cassiers et al. – realizes a shared version of a two-input AND gate – generic for any security order  $d$ . It is built based on a DOM-AND gadget and additionally refreshing one of the inputs. As a consequence, additional randomness has to be introduced into the design in order to guarantee SNI-conform refreshing. Here, the number of additional random bits compared to the DOM-AND depends on the number of shares, i.e., the security order  $d$ .

HPC2 [14]. In the same work, Cassiers et al. introduced HPC2, which, in contrast to HPC1, further reduces randomness requirements. It is based on blinding terms with fresh randomness depending on their cross domain and guaranteeing that no single glitch-extended probe can observe two terms involving the same randomness at the same time.

GHPC [31]. In their work, Knichel et al. presented a methodology for transforming any vectorial Boolean function  $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$  into a Hardware Private Circuit providing security and composability under the PINI notion in the glitch-extended probing model. As a result, GHPC gadgets enable, for example, the creation of a single gadget realizing entire SBOXes, while requiring only a low amount of fresh randomness, but are limited to the first security order.

GHPC<sub>LL</sub> [31]. In the same line of work, Knichel et al. additionally introduced GHPC<sub>LL</sub>, enabling the realization of any vectorial Boolean function in one clock cycle – at the cost of significantly higher randomness requirements. Similar to GHPC, GHPC<sub>LL</sub> is restricted to the first security order.

**4.2 Iterated Transition-Robust Hardware Private Circuits**

O-PINI1 [16]. O-PINI1 – recently introduced by Cassiers and Standaert – realizes a masked two-input AND gate that is provably composable under iterated transitions, i.e., when there exist a feedback loop in a design and one input of a gadget depends on the output of the same gadget, and which can be instantiated for arbitrary security orders  $d$ . Their construction is directly based on HPC2, only adding an additional refresh of its output. Note that this gadget is either glitch-robust or iterated transition-robust, but not both at the same time. In this work, we focus our consideration on the case where both, glitches and transitions, happen simultaneously, as this is the relevant case for hardware implementations.

O-PINI2 [16]. For the purpose of achieving trivial composability in the presence of iterated transitions *and* glitches, Cassiers and Standaert introduced O-PINI2 in the same work. In comparison to O-PINI1, it introduces an additional register stage at the end of the design, subsequent to the refresh of HPC2’s output, further restricting propagation of output probes and hence enabling trivial composition, even in the presence of both transitions and glitches.

### 4.3 Low-Latency Hardware Designs

CMS<sub>LL</sub> [35]. In this work, Molteni et al. adapted the original Consolidating Masking Schemes (CMS) multiplier to be effectively realized in a single clock cycle by adding pre-computed sums of random bits to the cross domains. Although the achieved design is glitch-robust SNI and not PINI – and hence linear operations cannot be trivially (share-wise) implemented – it is the only other existing composable gadget which realizes an AND operation within a single clock cycle for arbitrary security orders. Hence, we give a detailed comparison between this scheme and our construction later in this section.

LMDPL [42]. In their work, Sasdrich et al. presented a first-order secure, masked AES with a latency of a single cycle per round by utilizing the concept of LUT-based masked dual-rail pre-charge logic (LMDPL). Although the scheme can be seen as a gadget-based approach, the application of LMDPL gadgets requires circuit-specific signals, generated by a dedicated module called “masked table generator”. The authors themselves alleviated the scheme’s generality by stating that the underlying technique might not guarantee the same level of SCA resistance when used in ciphers with smaller/fewer SBOXes. As it is only restricted to the first security order and may offer lower levels of security when applied in other contexts, we omit a further, more detailed comparison to this design scheme.

To the best of our knowledge, other non-linear, composable hardware gadgets like DOM [20, 24] offer strictly worse latency than our newly proposed gadget. In [22], Gross et al. proposed an algorithmic-level hardware masking scheme to achieve low latency for arbitrary security orders which is rooted in the idea of skipping the compression layer of non-linear DOM gadgets whenever possible while ensuring that the sharing of inputs to every non-linear function are independent. Albeit its good results in terms of latency reduction, no formal security proof is given in the robust probing model and, as elaborated in [36], the scheme may suffer from composability issues for higher security orders when instantiated in different settings.

### 4.4 Comparison

A comparison of our designs with other state-of-the-art PINI gadgets is given in Table 1. Here, the latency is given as the number of register stages, and the required number of random bits is listed. Note that GHPC and GHPC<sub>LL</sub> are the only gadgets limited to the first security order. Similar to HPC2, our newly presented HPC3 has a constant latency regardless of the security order, while requiring only a single register stage instead of two. On the other hand, the number of required random bits for HPC3 is doubled compared to HPC2. We would like to highlight that due to the mandatory blinding of one share domain in order to derive the cross terms  $X^i \cdot Y^j$ , our construction achieves an optimal latency with respect to achieving trivial composability under the notion of PINI in the glitch-extended probing model. As a consequence, HPC3 allows the designer to trade double the randomness for half of the latency, or even combining HPC3 and HPC2 gadgets (as they are both composable under the PINI notion), providing significantly more flexibility with respect to adjusting the latency and randomness requirements of a masked implementation in different use cases.

**Table 1: Comparison of existing trivially composable Hardware Private Circuits**

Scheme	Latency	Randomness	Function	Ref.
HPC1	2	$d(d+1)/2 + r[d]^*$	AND	[14]
HPC2	2	$d(d+1)/2$	AND	[14]
GHPC <sup>†</sup>	2	$m$	$F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$	[31]
GHPC <sub>LL</sub> <sup>†</sup>	1	$2^n \cdot m$	$F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$	[31]
HPC3	1	$d(d+1)$	AND	[new]
O-PINI2 <sup>‡</sup>	3	$d(d+1)/2 + d$	AND	[16]
HPC3 <sup>+</sup> <sup>‡</sup>	2	$d(d+1) + d$	AND	[new]

\*  $r[d] = [1, 2, 4, 5, 7, 9, 11, 12, 15, 17]$  for  $d \leq 10$

<sup>†</sup> restricted to  $d = 1$

<sup>‡</sup> with robustness against iterative transitions+glitches

**Table 2: Comparison of single-cycle glitch-robust AND gadgets**

Scheme	Framework	$d$	Randomness	Ref.
CMS <sub>LL</sub>	SNI	1	2	[35]
		2	6	
		3	16	
		$\geq 4$	$2(d+1)^2$	
HPC3	PINI	1	2	[new]
		2	6	
		3	12	
		$\geq 4$	$d(d+1)$	

We further compare the existing transition-robust gadget with our newly introduced one. Applying the same methodology as described in [16] to our HPC3, we derived HPC3<sup>+</sup>. Similar to O-PINI2, our HPC3<sup>+</sup> construction offers trivial composability under the simultaneous presence of both, iterated transitions *and* glitches, while again trading one clock cycle less for  $d(d+1)/2$  additional fresh random bits.

In Table 2, a comparison between the required fresh masks of the aforementioned single-cycle robust-SNI gadget [35] and our HPC3 is given. While our design is already advantageous for  $d = 3$ , it needs less than half of their randomness requirements for  $d \geq 4$ . Another advantage stems from simpler composition conditions of PINI in comparison to SNI. As shown in [4, 14], trivial, i.e., share-wise, implementations of linear operations are not SNI. This possibly - depending on the circuit structure - demands for additional robust-SNI refresh gadgets to be inserted into the circuit, further increasing the introduced overhead. As the original composition approach by Barthe et al. [4] was over-conservative with respect to the insertion of refresh gadgets, Belaïd et al. introduced a new method in [10], drastically reducing the number of required refresh gadgets. They even utilize their method to generate a secured SNI-based AES implementation without any need for additional refresh gadgets.

Although this approach immensely reduces the number of necessary refresh gadgets in the SNI context, the overhead still has a higher dependency on the circuit's structure than in the context of PINI where each non-linear gate can simply be substituted by its gadget variant and the latency is hence simply determined by the maximum number of non-linear gates in any signal path.

To sum up, while there exist certain circuit structures where our approach will achieve similar results for low security orders when compared to CMS<sub>LL</sub>, the latency, the randomness requirements, and the area (additional registers, larger sources to generate randomness) of an arbitrary design (including all linear operations and layers of the cipher) composed of HPC3 gadgets will be favorable to using CMS<sub>LL</sub> in most of the cases - and for all cases when  $d \geq 3$ .

#### 4.5 Algorithmic-Level Masking vs. Composable Gadgets

Instead of utilizing composable gadgets, there exist several works aiming to derive highly optimized masked version for a certain cipher and design architecture by manually and carefully constructing each block/module in order to ensure security of the entire overarching design [42, 44, 45, 47, 47]. We refer to these approaches as *algorithmic-level* masking. The main advantage of these approaches is that they typically lead to a highly optimized implementation with respect to the introduced overhead. On the downside, these approaches result in implementations that are commonly restricted to a low (mostly the first or second) security order. Extending them to higher orders and translating them to other ciphers or design architectures is not trivially possible, or not possible at all. They further require a high expertise while the top-level designs' security cannot be easily proved in the formal  $d$ -probing model which is rooted in the fact that current formal verification tools like SILVER [30] and IronMask [8] are not able to cope with large circuits. Although these works typically present a leakage assessment of the resulting implementation, we would like to highlight that these leakage assessments are limited to making security statements about the implementation in a very specific setup, while the  $d$ -probing model aims to generalize the SCA resilience of an implementation.

Following a divide-and-conquer approach, composable gadgets, on the other hand, enable automated construction of masked circuits [29] by leveraging composable notions like PINI in order to ensure a provably secure top-level implementation, while typically coming at the cost of higher overheads. These schemes are not bounded to any cipher or design architecture but enable straightforward transformation of any circuit into its masked variant at any security order (with GHPC/GHPC<sub>LL</sub> being an exception restricted to the first order). In order to provide a fair comparison and as algorithmic-level masking and composable gadgets are conceptually different techniques, we omit any further benchmark comparison with algorithmic-level masking schemes in this work.

#### 4.6 Discussion on Latency-Optimized Logic Representations of Masked Circuits

In [14], Cassiers et al. described an optimized design strategy for small SBOXes using a SAT solver to find a Boolean representation such that one input to each HPC2 gadget in a later circuit stage is a linear combination of inputs to an earlier stage. Due to the

unbalance with respect to the input latency of HPC2, this leads to additional overall latency reduction. They showed, for example, that they can realize some 4-bit SBOXes utilizing only 3 register stages instead of trivially using 4 in a binary-tree multiplication of each monom in the respective coordinate functions. Despite the fact that this optimization approach is not practical for more complex functions, e.g., the AES SBOX, our HPC3 gadget still outperforms HPC2 latency-wise even in these cases where the SBOX representation is optimized in favor of HPC2. Note that only 2 clock cycles are necessary to realize a 4-bit SBOX in a trivial way. Hence, in favor of trivial composability, this optimization step is not required anymore when employing our HPC3 gadgets.

## 5 EXPERIMENTS AND EVALUATIONS

In order to give an overview of the performance figures of our constructed gadgets and the circuits composed of them, in this section, we present several case studies followed by experimental evaluation results.

### 5.1 Case Studies

Before we explain the details of our covered case studies, we refer to the recently introduced tool for automated generation of masked hardware implementations, AGEMA [29], which is publicly available through GitHub<sup>1</sup>. As given in Section 2.6, it receives the gate-level netlist of the unprotected implementation and generates a masked version of the same design using the specified gadgets. Although different processing methods are offered by AGEMA, we limit our comparisons to the *naive* approach which does not re-synthesize the given netlist and just replaces the gates with their corresponding gadgets. For the sake of comparability, we focus only on utilizing HPC2, O-PINI2, HPC3, and HPC3<sup>+</sup> gadgets which can all be instantiated for arbitrary security orders. To this end, we adopted the customized library of AGEMA and specified the gates of the aforementioned gadgets. We further constructed generic VHDL code of these gadgets (for all 2-input gates), where the desired security order  $d$  can be arbitrarily adjusted. For area comparison, we utilized Synopsys Design Compiler, together with the publicly-available standard library Nangate 45 for synthesizing the design.

*Small Circuits.* As the first case study, we compare single realizations of each of the considered AND gadgets, instantiated for the first three security orders  $d \in \{1, 2, 3\}$ . The corresponding results are shown in Table 3, clearly indicating the benefits of our constructed gadgets with respect to latency and area requirements. Our newly introduced HPC3 requires 20% less area compared to their counterpart HPC2 gadget. This advantage reaches even to around 55% in case of HPC3<sup>+</sup> compared to O-PINI2.

As a more realistic case study, we consider the 4-bit SBOX of the SKINNY-64 cipher [7]. The authors of [14] have provided a netlist for the SKINNY SBOX, which is optimized for HPC2 gadgets. The same netlist has been used for benchmarking purposes of AGEMA in [29]. Hence, we have taken the same netlist and constructed the SKINNY SBOX using our new gadgets. The results, which are shown in Table 4, reflect roughly the same conclusion as for the

<sup>1</sup><https://github.com/Chair-for-Security-Engineering/AGEMA>

**Table 3: Performance figures, 2-input AND.**

Scheme	Security [order]	Latency [cycle]	Rand. [bit]	Area [GE]	Ref.
HPC2	1	2	1	53	[14]
	2	2	3	156	
	3	2	6	311	
HPC3	1	1	2	43	[new]
	2	1	6	125	
	3	1	12	249	
O-PINI2	1	3	2	137	[16]
	2	3	5	301	
	3	3	9	529	
HPC3 <sup>+</sup>	1	2	3	69	[new]
	2	2	8	167	
	3	2	15	306	

**Table 4: Performance figures, Skinny SBOX [14] and AES SBOX [13].**

Scheme	Security [order]	Latency [cycle]	Rand. [bit]	Area [GE]	Ref.
<b>Skinny SBOX</b>					
HPC2	1	4	4	281	[14]
	2	4	12	730	
	3	4	24	1384	
HPC3	1	2	8	241	[new]
	2	2	24	606	
	3	2	48	1133	
<b>AES SBOX</b>					
HPC2	1	8	34	2189	[14]
	2	8	102	5923	
	3	8	204	11469	
HPC3	1	4	68	1849	[new]
	2	4	204	4855	
	3	4	408	9261	

2-input AND. The benefit of our gadgets with respect to latency is highly visible, but the area advantage is slightly mitigated, which is justified by the presence of the same XOR gadgets in all designs independent of the employed gadget family. More precisely, the application of our gadgets would still lead to a lower area overhead by a magnitude of 10%–20% while halving the latency. Note that for the SBOX and full-cipher case studies, we compare the results corresponding to only HPC2 and HPC3, since we believe that O-PINI2 and HPC3<sup>+</sup> are mainly relevant for the iterative designs shown in Figure 3 which we already compared in Table 3.

For the AES SBOX, we refer to [13], where a description for the inversion in  $GF(2^8)$  is given which needs a low number of cascaded 2-input gates. The same design has been used in the case studies of AGEMA as well. Hence, our AES SBOX case study is made based on this netlist, which has 4 layers of cascaded 2-input AND

**Table 5: Synthesis results, SKINNY-64-64 round-based encryption.**

Gadget	Pipel.	Security [order]	Area [GE]	Delay [ns]	Rand. [bits]	Latency [cycles]
unprotected	-	0	1494	0.52	-	33
HPC2	✓ ✓ ✓	1	6895	0.55	64	165
		2	15193	0.61	192	165
		3	26777	0.65	384	165
		1	20210	0.53	64	165
		2	36147	0.59	192	165
		3	56096	0.63	384	165
HPC3	✓ ✓ ✓	1	6467	1.06	128	99
		2	13517	1.18	384	99
		3	23171	1.19	768	99
		1	13462	0.59	128	99
		2	23956	0.64	384	99
		3	37051	0.66	768	99

**Table 6: Synthesis results, AES-128 round-based encryption.**

Gadget	Pipel.	Security [order]	Area [GE]	Delay [ns]	Rand. [bits]	Latency [cycles]
unprotected	-	0	9906	1.85	-	11
HPC2	✓ ✓ ✓	1	52597	2.04	680	99
		2	131631	2.39	2040	99
		3	246924	2.53	4080	99
		1	161440	0.82	680	99
		2	305274	0.89	2040	99
		3	492077	0.93	4080	99
HPC3	✓ ✓ ✓	1	44581	2.11	1360	55
		2	108476	2.34	4080	55
		3	200307	2.33	8160	55
		1	94450	0.79	1360	55
		2	182883	0.83	4080	55
		3	299013	0.86	8160	55

gates. Since our HPC3 gadgets make use of only one register stage, this naturally leads to 4 clock cycles latency for the entire masked AES SBOX at any arbitrary order. The corresponding results are illustrated in Table 4, indicating 15%–20% area reduction in addition to a naturally lower latency compared to the equivalent state-of-the-art designs. As a side note, in all results presented above, we have not considered extra registers required to construct pipeline designs, which is a feature of AGEMA.

*Full Ciphers.* In order to have a better overview on the benefit and overhead of our constructed gadgets, we took a selection of full-cipher designs used by AGEMA as the case study, including the SKINNY-64-64 round-based encryption and the AES-128 round-based encryption function. The corresponding results are given in Table 5 and Table 6.

We provided the comparative results for the first three security orders  $d \in \{1, 2, 3\}$  as well as for the pipeline and non-pipeline designs. The pipeline designs, which naturally have higher area footprints, can process multiple sequentially-given inputs. For example, we refer to Table 5, where HPC2 designs have 165 clock

cycles latency and HPC3 designs 99 clock cycles. The non-pipeline designs, receive a single plaintext and key and perform the encryption after 165 (resp. 99) clock cycles, what the unprotected design does in 33 clock cycles. The HPC2 pipeline design can process 4 individual plaintext-key pairs in 165 clock cycles, and the HPC3 pipeline design 2 individual inputs in those 99 clock cycles.

The results are indeed along the same line as those given for SBOX case studies. More precisely, the area overhead and latency of the circuits made by HPC3 gadgets are less than those made by HPC2 gadgets while demanding for double amount of fresh randomness. The area advantage becomes more obvious at higher orders – particularly for pipeline designs. We should highlight that, in contrast to the SBOX cases studies, the latency of HPC3 circuits is not exactly half of the latency of HPC2 circuits since registers already exist in the unprotected designs. AGEMA models the given unprotected design as a Mealy machine made by a single-stage register and a fully combinational circuit whose inputs consists the circuit's primary input and the output of the registers provided by a feedback loop (for more details see [29, § 2.6]). The role of HPC2, HPC3 or any other gadget lies in how the combinational circuit is converted into a secure one. Therefore, the register stage of the Mealy machine stays as it is; just being extended based on the number shares, i.e., the desired security order  $d$ . As a result, if we denote the latency of the unprotected circuit by  $\eta$ , the latency of the converted circuit becomes  $(l + 1)\eta$  cycles, where  $l$  stands for the latency of the combinational circuit realized by HPC2 gadgets. This means that the latency of the same circuit implemented by HPC3 gadgets becomes  $(l/2 + 1)\eta$  clock cycles which can also be seen in Tables 5 and 6. It is worth to highlight that all our HDL designs of the gadgets and the case studies are provided in the GitHub: <https://github.com/Chair-for-Security-Engineering/HPC3>. Additional case studies can also be found in the extended version of this paper [28].

## 5.2 Leakage Assessment

Although (robust) probing security is implied by our gadget's conformity to the PINI notion, we conducted further analysis for the purpose of presenting a complete work and showing our methodologies final practical security. By means of SILVER [30], we have verified the security of the constructed SBOXes (reported in Section 5.1) under the glitch-extended probing model. In short, all our constructions are reported secure up to the desired security order. However, evaluation of larger designs, e.g., a cipher round, is out of the feasibility limits of such verification tools. Therefore, the remaining choice to evaluate a full cipher implementation is to conduct experimental analyses. To this end, we employed a Spartan-6 FPGA-based evaluation platform (SAKURA-G [41]), and collected power consumption traces of different designs to conduct various fixed-versus-random t-tests [6].

*Setup.* We monitored the output of the embedded AC-amplifier of the SAKURA-G which senses the voltage drop over a  $1\ \Omega$  shunt resistor placed on the VDD path of the target FPGA, and collected power consumption traces by sampling such an amplified signal at a sampling frequency of 500 MS/s. During this time, the underlying design under test was supplied by a stable and jitter-free clock at a frequency of 6 MHz.

For the sake of comparability with the state of the art, we choose the SKINNY-64-64 encryption function for experimental analysis, since the same has been used in [29]. This choice has indeed neither an effect on the validity of our experiments nor on the security of our constructions. An obvious choice is to examine the AES implementations. However, the third-order round-based AES can hardly fit into our FPGA setup, while all variants of SKINNY-64-64 easily fit.

Nevertheless, we have taken the first- to third-order pipeline designs reported in Table 5. In short, our designs require 99 clock cycles to accomplish the encryption, in contrast to 33 clock cycles for the unprotected design and 165 clock cycles for the designs protected by HPC2 gadgets. Further, our designs require 128, 384, and 768 fresh mask bits, for first- to third-order security respectively, which should be updated at every clock cycle (the same when using HPC3 gadgets).

When measuring the power consumption of our designs, we made sure to cover all 99 clock cycles of each entire encryption process. In order to supply the required fresh masks, we made use of the FPGA-optimized construction illustrated in [34], which realizes an individual 31-bit Linear Feedback Shift Register (LFSR)<sup>2</sup> for every fresh mask bit, seeded randomly at the power-up of the device and updated at every clock cycle.

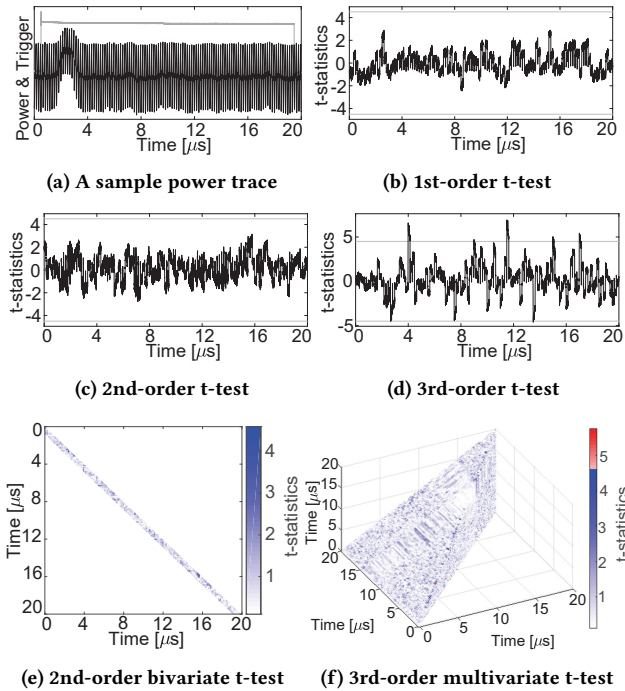
*Results.* To analyze the implementations, we conducted various forms of a fixed-versus-random t-test, commonly referred to as TVLA [6, 43]. It is a well-known leakage assessment technique that is able to detect SCA leakage in measurements collected from cryptographic implementations by examining whether the leakages associated to two groups are distinguishable; one group with a fixed plaintext and the other one with randomly-chosen plaintexts, while a constant and identical key is used in both groups. In all cases, the entire inputs (plaintext + key) are given to the circuit in a masked form with respect to the security order of the underlying design.

In the following we will give a detailed exploration of the leakage assessment for our second-order secure design ( $d=2$ ). Here, we first performed the ordinary t-test on each sample point individually (i.e., univariate), whose corresponding results – shown in Figure 5b – confirm its first-order security. For higher-order t-tests, we made the traces mean-free (for each group of fixed and random individually). Afterwards, each mean-free sample point is squared (resp. cubed), before calculating the t-statistics for univariate second-order (resp. univariate third-order) t-tests (see Figure 5c and Figure 5d). The results are as expected, i.e., show no leakage for the second order.

For the bivariate second-order t-tests, we performed an individual t-test for each combination of every two possible sample points by multiplying the corresponding mean-free power values. In our experiments, each power trace contains 10 000 sample points, which translates to  $10\ 000 \times 10\ 000 / 2 = 50\ 000\ 000$  individual t-tests, hence a very time-intensive computation even using large CPU clusters. Since our constructions utilize fresh masks (updated every clock cycle), any possible bivariate leakage is expected to be present for a combination between sample points in close proximity. Therefore, we limited our bivariate analysis to sample points within a distance of at most five clock cycles. This strongly reduces the amount of computations and allows us to accomplish the evaluations in a

<sup>2</sup>With feedback polynomial  $x^{31} + x^{28} + 1$ .





**Figure 5: Experimental analysis of SKINNY-64 encryption round-based design, masked with second-order HPC3 gadgets using 100 million traces.**

reasonable time frame. As expected, the corresponding results – shown in Figure 5e – confirm the non-existence of second-order bivariate leakages.

For the third-order multivariate analysis, the processing is even more complex. If we limit the maximum distance between the sample points to, e.g., five clock cycles, the number of possible combinations of three sample points is way above the feasibility threshold. In order to find an alternative approach, we refer to Figure 5d and common knowledge indicating that the amount of leakage (in power traces) associated to a clock cycle is approximately the same for the entire clock cycle. Therefore, an appropriate sample point per clock cycle should suffice for such analyses. Indeed, this is a known concept referred to as memory effect in power consumption measurements due to the low-pass filter inherently built by the components involved in the measurement setup, e.g., the shunt resistor, the chip package, and the Printed Circuit Board (PCB) [37]. Therefore, we down-sampled the traces by taking a sample point for each clock cycle (carefully selected at the middle of the cycle). Note that such a down sampling and restricting the multivariate analysis to a small period of time has been done in the state of the art as well [11, 18, 46, 49]. The result of this analysis is shown by a 3D pyramid in Figure 5f, indicating some tuples (of three points) whose combination (mean-free product) leads to detectable leakage. Note that third-order univariate and multivariate leakages are expected in case of this second-order-secure design; we just showed the detailed results of such analyses as a proof of functionality of our setup.

The same procedure has been performed on the first - and third-order designs ( $d=1$  and  $d=3$  respectively). For the sake of brevity and as the results are as expected, i.e., no leakage can be detected up to the  $d$ -th order, we leave out any detailed depiction here. The interested reader can find these results in the extended version of this paper [28].

## 6 CONCLUSIONS

In this work, we presented HPC3, a low-latency Hardware Private Circuit that is trivially composable under the PINI notion in the glitch-extended robust probing model. To the best of our knowledge, this is the first PINI-based hardware gadget that realizes a masked AND-gate in a single clock cycle for any arbitrary security order. We further gave the algorithm for its construction and formally proved its conformity to the PINI notion in the glitch-extended  $d$ -probing model. We should stress that the achieved latency of a single clock cycle is the lowest bound for trivial composability of an AND gadget under the notion of PINI. This is due to the necessary computation of cross terms, i.e., multiplication of shares from different share domains. Hence, HPC3 is the first proposed gadget achieving this lowest bound for arbitrary security orders. We additionally presented HPC3<sup>+</sup>, a masked AND-gate that is trivially composable under the presence of both, iterated transitions and glitches and can be instantiated for arbitrary security orders  $d$ . Moreover, we practically verified the security of our constructions by means of various case studies and leakage assessments.

Compared to existing state-of-the-art Hardware Private Circuits, HPC3 enables the designers to make use of more fresh randomness to halve the latency while maintaining trivial composability (PINI) in the glitch-extended probing model. This offers a significant speedup of cryptographic implementations for different use cases and paving the ground for secure, low-latency applications. A similar trade-off can be made with HPC3<sup>+</sup>, which – compared to OPINI-2 – enables the designers to omit one register stage and hence reducing the design’s latency by a factor of 2/3.

*Future Works.* It remains an interesting open topic, if such a trade-off between the demand for fresh randomness and latency can be also found for larger gates, i.e., with more inputs than two. Constructing composable and generic hardware gadgets for 3- or 4-input non-linear gates can highly increase the efficiency of masked implementation with respect to latency. Further, there still exists yet no detailed cost function to examine the overhead introduced by different metrics. For example, it is not clear at which point area consumption required to generate additional fresh randomness compensates the area gain by requiring fewer register stages. It would be hence beneficial for the research community to have detailed and realistic cost functions for these factors.

## ACKNOWLEDGMENTS

The work described in this paper has been supported in part by the Deutsche Forschungs-gemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy- EXC 2092 CASA - 390781972 and through the projects 35264177 SAUBER and 393207943 GreenSec.

## REFERENCES

- [1] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. 2018. Private Circuits: A Modular Approach. In *CRYPTO 2018 (Lecture Notes in Computer Science)*, Vol. 10993. Springer, 427–455.
- [2] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. 2019. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In *ESORICS 2019 (Lecture Notes in Computer Science)*, Vol. 11735. Springer, 300–318.
- [3] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. 2015. Verified Proofs of Higher-Order Masking. In *EUROCRYPT 2015 (Lecture Notes in Computer Science)*, Vol. 9056. Springer, 457–485.
- [4] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. 2016. Strong Non-Interference and Type-Directed Higher-Order Masking. In *CCS 2016*. ACM, 116–129.
- [5] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. 2017. Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. In *EUROCRYPT 2017 (Lecture Notes in Computer Science)*, Vol. 10210. Springer, 535–566.
- [6] G. Becker, J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leiserson, M. Marson, P. Rohatgi, and S. Saab. 2013. Test vector leakage assessment (TVLA) methodology in practice. In *International Cryptographic Module Conference*.
- [7] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. 2016. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In *CRYPTO 2016 (Lecture Notes in Computer Science)*, Vol. 9815. Springer, 123–153.
- [8] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. 2022. IronMask: Versatile Verification of Masking Security. In *IEEE SP 2022*. IEEE.
- [9] Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. 2020. Tornado: Automatic Generation of Probing-Secure Masked Bitsliced Implementations. In *EUROCRYPT 2020 (Lecture Notes in Computer Science)*, Vol. 12107. Springer, 311–341.
- [10] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. 2018. Tight Private Circuits: Achieving Probing Security with the Least Refreshing. In *ASIACRYPT 2018 (Lecture Notes in Computer Science)*, Vol. 11273. Springer, 343–372.
- [11] Tim Beyne, Siemen Dhooghe, Amir Moradi, and Ain Rezaei Shahmirzadi. 2022. Cryptanalysis of Efficient Masked Ciphers: Applications to Low Latency. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 1 (2022), 679–721.
- [12] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. 2018. Formal Verification of Masked Hardware Implementations in the Presence of Glitches. In *EUROCRYPT 2018 (Lecture Notes in Computer Science)*, Vol. 10821. Springer, 321–353.
- [13] Joan Boyar and René Peralta. 2012. A Small Depth-16 Circuit for the AES S-Box. In *Information Security and Privacy Conference, SEC 2012 (IFIP)*, Vol. 376. Springer, 287–298.
- [14] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. 2021. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. Computers* 70, 10 (2021), 1677–1690.
- [15] Gaëtan Cassiers and François-Xavier Standaert. 2020. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Inf. Forensics Secur.* 15 (2020), 2542–2555.
- [16] Gaëtan Cassiers and François-Xavier Standaert. 2021. Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 2 (2021), 136–158.
- [17] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. 1999. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO 1999 (Lecture Notes in Computer Science)*, Vol. 1666. Springer, 398–412.
- [18] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventsislav Nikov, and Vincent Rijmen. 2016. Masking AES with  $d+1$  Shares in Hardware. In *CHES 2016 (Lecture Notes in Computer Science)*, Vol. 9813. Springer, 194–212.
- [19] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. 2014. Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In *EUROCRYPT 2014 (Lecture Notes in Computer Science)*, Vol. 8441. Springer, 423–440.
- [20] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. 2018. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 89–120.
- [21] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. 2001. Electromagnetic Analysis: Concrete Results. In *CHES 2001 (Lecture Notes in Computer Science)*, Vol. 2162. Springer, 251–261.
- [22] Hannes Groß, Rinat Iusupov, and Roderick Bloem. 2018. Generic Low-Latency Masking in Hardware. *TCHES 2018* 2018, 2 (2018), 1–21.
- [23] Hannes Groß and Stefan Mangard. 2018. A unified masking approach. *J. Cryptogr. Eng.* 8, 2 (2018), 109–124.
- [24] Hannes Groß, Stefan Mangard, and Thomas Korak. 2016. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *TIS@CCS 2016*. ACM, 3.
- [25] Hannes Groß, Stefan Mangard, and Thomas Korak. 2017. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In *CT-RSA 2017 (Lecture Notes in Computer Science)*, Vol. 10159. Springer, 95–112.
- [26] Michael Hutter and Jörn-Marc Schmidt. 2013. The Temperature Side Channel and Heating Fault Attacks. In *CARDIS 2013 (Lecture Notes in Computer Science)*, Vol. 8419. Springer, 219–235.
- [27] Yuval Ishai, Amit Sahai, and David A. Wagner. 2003. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003 (Lecture Notes in Computer Science)*, Vol. 2729. Springer, 463–481.
- [28] David Knichel and Amir Moradi. 2022. Low-Latency Hardware Private Circuits. Cryptology ePrint Archive, Paper 2022/507. (2022). <https://eprint.iacr.org/2022/507>
- [29] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. 2022. Automated Generation of Masked Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 1 (2022), 589–629.
- [30] David Knichel, Pascal Sasdrich, and Amir Moradi. 2020. SILVER - Statistical Independence and Leakage Verification. In *ASIACRYPT 2020 (Lecture Notes in Computer Science)*, Vol. 12491. Springer, 787–816.
- [31] David Knichel, Pascal Sasdrich, and Amir Moradi. 2022. Generic Hardware Private Circuits Towards Automated Generation of Composable Secure Gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 1 (2022), 323–344.
- [32] Paul C. Kocher. 1996. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO 1996 (Lecture Notes in Computer Science)*, Vol. 1109. Springer, 104–113.
- [33] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *CRYPTO 1999 (Lecture Notes in Computer Science)*, Vol. 1666. Springer, 388–397.
- [34] Lauren De Meyer, Amir Moradi, and Felix Wegener. 2018. Spin Me Right Round Rotational Symmetry for FPGA-Specific AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 596–626.
- [35] Maria Chiara Moltteni, Jürgen Pulkus, and Vittorio Zaccaria. 2022. On robust strong-non-interferent low-latency multiplications. *IET Inf. Secur.* 16, 2 (2022), 127–132.
- [36] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. 2019. Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 2 (2019), 256–292.
- [37] Amir Moradi and Oliver Mischke. 2013. On the Simplicity of Converting Leakages from Multivariate to Univariate - (Case Study of a Glitch-Resistant Masking Scheme). In *CHES 2013 (Lecture Notes in Computer Science)*, Vol. 8086. 1–20.
- [38] Svetla Nikova, Vincent Rijmen, and Martin Schläpfer. 2011. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptol.* 24, 2 (2011), 292–321.
- [39] Emmanuel Prouff and Matthieu Rivain. 2013. Masking against Side-Channel Attacks: A Formal Security Proof. In *EUROCRYPT 2013 (Lecture Notes in Computer Science)*, Vol. 7881. Springer, 142–159.
- [40] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. 2015. Consolidating Masking Schemes. In *CRYPTO 2015 (Lecture Notes in Computer Science)*, Vol. 9215. Springer, 764–783.
- [41] SAKURA. 2022. Side-channel Attack User Reference Architecture. <http://satoh.cs.ucc.ac.jp/SAKURA/index.html>. (2022).
- [42] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. 2020. Low-Latency Hardware Masking with Application to AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020, 2 (2020), 300–326.
- [43] Tobias Schneider and Amir Moradi. 2015. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In *CHES 2015 (Lecture Notes in Computer Science)*, Vol. 9293. Springer, 495–513.
- [44] Ain Rezaei Shahmirzadi, Dusan Bozilov, and Amir Moradi. 2021. New First-Order Secure AES Performance Records. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 2 (2021), 304–327.
- [45] Ain Rezaei Shahmirzadi and Amir Moradi. 2021. Re-Consolidating First-Order Masking Schemes Nullifying Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 1 (2021), 305–342.
- [46] Ain Rezaei Shahmirzadi and Amir Moradi. 2021. Second-Order SCA Security with almost no Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 3 (2021), 708–755.
- [47] Takeshi Sugawara. 2019. 3-Share Threshold Implementation of AES S-box without Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 1 (2019), 123–145.
- [48] Elena Trichina. 2003. Combinational Logic Design for AES SubByte Transformation on Masked Data. *IACR Cryptol. ePrint Arch.* 2003 (2003), 236.
- [49] Sara Zarei, Ain Rezaei Shahmirzadi, Hadi Soleimany, Raziye Salarifard, and Amir Moradi. 2021. Low-Latency Keccak at any Arbitrary Order. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 4 (2021), 388–411.

# Chapter 6

## Methodologies and Tooling for Automated Masking

*In this chapter, we present a peer-reviewed publication published in the context of this thesis that relates to methodologies and tooling for automated masking. In particular, we present one paper published in in the IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES).*

### Contents of this Chapter

---

6.1 Automated Generation of Masked Hardware . . . . .	203
---	-----

---

## 6.1 Automated Generation of Masked Hardware

### Publication Data

David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. Automated Generation of Masked Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):589–629, 2022

*This work is reproduced here with permission. This work is licensed under a Creative Commons Attribution 4.0 International License. Copyright is held by the authors. The author of this thesis is also an author of this research paper.*

**Content.** In our work, we introduce a tool for the automated generation of masking, simply based on the unprotected circuit. AGEMA offers different options for preprocessing the circuit before certain functional blocks of the design are replaced by their gadget counterpart. As gadgets introduce individual latency into the design, either pipeline registers are inserted into the design or clock gating is applied in order to guarantee proper functionality. For AGEMA’s trivial approach, atomic logic gates – like AND, OR – are simply replaced by their corresponding gadget equivalent. Next to introducing AGEMA as a fully functional tool for automated masking of hardware with many options to find trade-offs between area, latency and area of the resulting design, we perform several case studies and practical leakage assessments.

**Contribution.** The author contributed to developing and supplying gadget construction and the methodology for guaranteeing security under composition. Furthermore, the author was involved in the writing of this work. The author likes to thank all co-author for their contribution.



# Automated Generation of Masked Hardware

David Knichel\*<sup></sup>, Amir Moradi\*<sup></sup>, Nicolai Müller\*<sup></sup> and  
Pascal Sasdrich\*<sup></sup>

Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany  
[firstname.lastname@rub.de](mailto:firstname.lastname@rub.de)

**Abstract.** Masking has been recognized as a sound and secure countermeasure for cryptographic implementations, protecting against physical side-channel attacks. Even though many different masking schemes have been presented over time, design and implementation of protected cryptographic Integrated Circuits (ICs) remains a challenging task. More specifically, correct and efficient implementation usually requires manual interactions accompanied by longstanding experience in hardware design and physical security. To this end, design and implementation of masked hardware often proves to be an error-prone task for engineers and practitioners. As a result, our novel tool for *automated generation of masked hardware* (AGEMA) allows even inexperienced engineers and hardware designers to create secure and efficient masked cryptographic circuits originating from an unprotected design. More precisely, exploiting the concepts of *Probe-Isolating Non-Interference (PINI)* for secure composition of masked circuits, our tool provides various processing techniques to transform an unprotected design into a secure one, eventually accelerating and safeguarding the process of masking cryptographic hardware. Ultimately, we evaluate our tool in several case studies, emphasizing different trade-offs for the transformation techniques with respect to common performance metrics, such as *latency*, *area*, and *randomness*.

**Keywords:** Side-Channel Analysis · Masking · Hardware · Composable Gadget

## 1 Introduction

Side-Channel Analysis (SCA) has not lost any of its topicality and remains a major threat to security-critical implementations, even after more than two decades of intensive research since its seminal description. In the wake of this lasting discovery [Koc96, KJJ99], it has been admittedly recognized that secure implementation of cryptographic algorithms is a challenging task, given that an adversary can observe and measure physical effects in order to infer sensitive information during execution. Examples include timing [Koc96], power consumption [KJJ99], electromagnetic (EM) radiations [GMO01], or temperature and heat dissipation [HS13]. However, in the course of time, different classes of counteractive measures have emerged amongst which *masking* [CJRR99], based on concepts of *secret sharing*, prevails due to its formal and sound security foundation.

Over the last years, many different hardware masking variants and schemes have been proposed [ISW03, NRR06, RBN<sup>+</sup>15, GMK17, GM18], constantly improving efficiency and security. Unfortunately, experience has shown that new schemes often have a short retention time, mostly due to inaccuracies and design flaws [MMSS19]. However, even for schemes that stand the test of time, correct and secure implementation remains an enormous engineering challenge. As a matter of fact, even with longstanding experience

---

\*Authors list in alphabetical order; see <https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf>

and expertise in hardware security and design of masked hardware, correct physical instantiation of masking schemes is a delicate and error-prone task. Evidently, unconsidered and unintentional physical effects, e.g., *glitches* [MPG05], *transitions* [CGP<sup>+</sup>12, BGG<sup>+</sup>14], or *coupling* [CBG<sup>+</sup>17], and implementation defects due to architectural conditions, e.g., *parallelism* [BDF<sup>+</sup>17] or *pipelining* [CGD18], can render a theoretically secure scheme practically insecure.

As a consequence, a new line of research emerged, investigating the masking of atomic and reusable components, often considered as *gadgets* in literature, to limit the engineering complexity and error susceptibility [RBN<sup>+</sup>15, GMK16, GM18, GIB18, BBD<sup>+</sup>15, BBD<sup>+</sup>16, CS20]. In this regard, a great deal of attention has been devoted to the construction of secure gadgets for basic non-linear operations (e.g., AND), allowing to efficiently mask any digital logic circuit given its AND-XOR representation. However, the continuous progress in this domain is inevitably associated with fundamental research and advancements in the realm of formal security definitions and adversary models. More specifically, the formal and abstract Ishai-Sahai-Wagner (ISW)  $d$ -probing model [ISW03] is consulted prevalently to reason about security of masked circuits in the presence of side-channel adversaries.

Unfortunately, research has shown that security in this simple model does not imply secure composition of gadgets [CPRR13]. More precisely, composition of standalone-secure sub-circuits does not necessarily lead to a secure circuit. As a consequence, advanced security notions and properties are essential to reason about the *composability* of masked gadgets. In a first attempt, Barthe et al. [BBD<sup>+</sup>15] introduced the notion of Non-Interference (NI), allowing to verify the composability of gadgets based on the concept of *perfect simulation* of joint probability distributions. However, due to disregarded effects which later became known as *probe propagation* [CS20], the notion of NI is deficient and has been complemented by the notion of Strong Non-Interference (SNI) shortly afterwards [BBD<sup>+</sup>16]. Most recently, Cassiers and Standaert [CS20] introduced the notion of Probe-Isolating Non-Interference (PINI) enabling more efficient compositions with respect to *multi-input*, *multi-output* gadgets and *trivially secure* linear operations.

Now, provided with such sound and formal security and composability notions, hardware designers are able to construct secure circuits more easily. However, transforming an entirely unprotected design into a secure circuits remains a complicated and mostly manual process, even when endowed with an adequate set of secure and composable gadgets.

**Contributions.** In this work, we present our novel, and open-source, software tool for *automated generation of masked hardware* (AGEMA), enabling engineers and hardware designers of any level of expertise to easily generate masked hardware circuits starting from a simple but unprotected design. Utilizing different methods for processing the given netlist<sup>1</sup> of a design and supporting arbitrary masked gadgets, AGEMA offers high flexibility with respect to the security level (i.e., order of the underlying masking scheme), required fresh randomness, latency, and area overhead and consequently gives designers the ability to configure AGEMA to their particular needs. Exploiting the essential security notions for secure composability, the final designs are provably secure (under PINI notion) and hence are formally verified. They are further free of heuristics, and can mitigate implementation defects and design mistakes. As a consequence, our tool facilitates and accelerates the process of masking digital logic circuits while in the same vein, the quality and security of the resulting designs are increased.

We should highlight that, up to now, various tools have been developed to help generating masked software implementations. The examples include [BRB<sup>+</sup>11, BRN<sup>+</sup>15] and the recently-introduced ones Tornado [BDM<sup>+</sup>20] and Rosita [SSB<sup>+</sup>21]. Further, fullVerif [CGLS21] is a tool to analyze security of masked circuits at the composition

<sup>1</sup>In digital circuit design, a netlist is a description of the connectivity of a circuit. In its simplest form, a netlist consists of a list of the cells in a circuit and a list of the wires they are connected to.

level. It receives a masked implementation which is built by annotated PINI gadgets, and examines whether the connection between PINI gadgets is in compliance with the PINI definitions and requirements, i.e., checking if designers made any mistakes. In short, it does not build any secure circuit but only verifies correct composition. To the best of our knowledge, AGEMA is the only tool which is dedicated to generation of masked hardware implementations, and security of its generated circuits are based on the PINI security notion which guarantees to maintain the security through composition.

**Outline.** We start by summarizing our notations and all necessary theoretical concepts in [Section 2](#), before elaborating the fundamental methodology of AGEMA in [Section 3](#). This includes a brief summary of existing types of Hardware Private Circuit (HPC) and a detailed explanation for the supported processing and transformation methods of the original netlist. In [Section 4](#), we examine AGEMA with an extensive list of case studies on a broad variety of hardware implementations of different block ciphers and compare the generated masked designs with respect to common performance metrics such as required fresh randomness, latency degradation, and area overhead. Before we conclude our work, we provide reasoning behind the security of HPC circuits constructed by AGEMA in [Section 5](#), and give result of experimental analyses, i.e., Test Vector Leakage Assessment (TVLA), on some exemplary designs as outcome of the application of AGEMA.

## 2 Basics

### 2.1 Notations

We denote functions using sans-serif fonts, e.g.,  $f$  for Boolean functions and  $F$  for vectorial Boolean functions. Next, we denote single-bit random variables in  $\mathbb{F}_2$  by lower case letters like  $x$ , and vectors by uppercase letters  $X$  while sets of random variables are given in bold  $\mathbf{X}$ . Further, we use subscripts like  $x_i$  to indicate elements within a vector or a set while superscripts are used to denote (randomized) shares of random variables, e.g.,  $X^j$ . As a special case, the set of all shares of each random variable in  $\mathbf{X}$  is denoted as  $Sh(\mathbf{X})$ .

### 2.2 Boolean Masking

Masking is based on secret sharing and has proven to be well suited for hardware implementations as a countermeasure against side-channel attacks. In *Boolean masking*, a sensitive variable  $X \in \mathbb{F}_n$  is split into  $s \geq 2$  randomized shares  $(X^0, X^1, \dots, X^{s-1}) \in \mathbb{F}_n^s$ , such that  $X = \bigoplus_{i=0}^{s-1} X^i$ . Usually this sharing is initially achieved by sampling  $X^i \xleftarrow{\$} \mathbb{F}_n$  for all  $0 \leq i < s-1$  and calculating  $X^{s-1} = (\bigoplus_{i=0}^{s-2} X^i) \oplus X$ . Eventually, instead of performing logic operations on the sensitive value  $X$ , they will be performed on the (randomized) shared representation of  $X$ , i.e.,  $X^0, X^1, \dots$ , and  $X^{s-1}$ .

### 2.3 Probing Security

In order to abstract and formalize the behavior of a masked circuit and the adversarial capabilities to extract information from the underlying circuit, several models have been introduced over time, aiming to achieve different trade-offs between simplicity and accuracy. In the  $d$ -probing model, firstly introduced by Ishai et al. in [\[ISW03\]](#), the adversary is granted the ability to observe the distribution over up to  $d$  wires of a given circuit. To achieve  $d$ -probing security for a masked circuit, any adversary in conformity with this model should not be able to learn anything about the processed sensitive value  $X$ .

**Definition 1.** A masked circuit  $C$  is said to achieve  $d$ -probing security iff every (joint) distribution over up to  $d$  wires is statistically independent of any sensitive value  $X$ .

**Definition 1** directly implies that splitting any sensitive variable into at least  $d + 1$  shares is necessary to achieve  $d$ -probing security. In the context of masking,  $d$  is also referred to as the *security order* of a given masked circuit.

**Robust Probing Model and Glitch-Extended Probes.** Since the traditional probing model is limited to software implementations due to its inability to capture physical defaults occurring in hardware implementations, e.g., *transitions* (memory recombinations), *glitches* (combinational recombinations), or *coupling* (routing recombinations), the *robust* probing model was introduced in [FGP<sup>+</sup>18], aiming to consider and model these defaults accurately while being sufficiently simple in order to enable efficient verification of masked designs. In contrast to the traditional probing model, where the value of a wire is always assumed stable during evaluation and where no dedicated synchronization elements, i.e., registers, exist, the robust probing model loosens this assumption by introducing registers and so-called extended probes. Here, a single probe can be extended to additionally capture leakage caused by physical defaults like data transitions at registers, glitches in combinational logic, and coupling of adjacent wires.

In particular, *glitches* are switching activities of wires caused by different delays of signals contributing to their intended values. These glitches enable a probe on a single wire to observe not only the field element of its driving gate, but possibly a recombination of signals contributing to its combinatorial value. Hence, in order to capture these effects, *glitch-extended* probes were introduced. Here, a single probe on a wire is assumed to capture the leakage of the joint distribution over every stable signal contributing to the calculation of the probed wire. As a result, given the glitch-extended probing model, every probe on a wire is replaced by the set of probes placed on the register outputs and primary inputs that contribute to the observed wire, i.e., there exists a path from a stable source to the current probe position.

## 2.4 Composable Masking Schemes

Especially for higher security orders  $d$  and more complex functions, it is hard to find efficient masked representations for circuits to become provably  $d$ -probing secure, as the number of possible probe combinations increases with the security order and the complexity of a circuit. Following a divide-and-conquer approach, composable gadgets were introduced as a remedy to directly derive masked representations of large functions. Composable gadgets are masked circuits realizing small and atomic logic functions, like a simple AND or OR gate. Fundamentally, these gadgets fulfill certain properties that imply probing security when composed to construct a larger circuit. This way, the problem of finding secure masked realizations of large functions is reduced to the task of finding composable gadgets realizing small functions with certain properties. In other words, in contrary to what has been observed in [MMSS19], the gadgets should be designed in such a way that their composition does not lead to any security degradation.

**Probe Propagation and Composability Notions.** To understand favorable properties for gadgets in order to achieve secure composability, we explain the concept of *probe propagation*, which was firstly introduced in [CS20] and defines the information a probe can access and how this access to information is propagated throughout the circuit. Generally, a (glitch-extended) probe is said to *propagate* into a wire if this wire is needed to perfectly simulate each observation of the probe, i.e., in order to compute the underlying probability distribution. Now, to achieve composability of a gadget, propagation of internal probes and output probes needs to be restricted to a subset of the input wires of the gadgets. These constraints on gadget level have to guarantee that all possible probes in a composed circuit only propagate in a subset of the initial sharing of an input value and



not into all of them. After Non-Interference (NI) [BBD<sup>+</sup>15] was proven to be insufficient to offer composability, Strong Non-Interference (SNI) [BBD<sup>+</sup>16] was proposed which further restricts probe propagation and was originally restricted to single-output gadgets. In [CS20], Cassiers and Standaert showed that the scope of the original definition of SNI can be extended to cover multiple-output gadgets as well, but at the same time unveiled issues of SNI with respect to the extent of required entropy and circuit area. Eventually, Probe-Isolating Non-Interference (PINI) was introduced in the same work as an elegant way to guarantee composability at any security order. Similar to Domain Oriented Masking (DOM) [GMK16], share domains were introduced and any probe was restricted to only propagate within its own share domain, enabling trivial implementation of linear functions on the one hand and direct composition of gadgets on the other.

## 2.5 Formal Verification

Several tools have been published for the purpose of formally verifying the security and composability characteristics of masked hardware circuits [BGI<sup>+</sup>18a, BGI<sup>+</sup>18b, CGLS21, BBC<sup>+</sup>19, KSM20]. They all support different varieties of security and composability notions while working on different abstraction levels. We choose SILVER [KSM20] for performing all verification in this work, due to its unique support of checking composability under the PINI notion in the glitch-extended probing model.

## 2.6 Combinational and Sequential Circuits

*Combinational circuits* are digital circuits where the output is a pure function of the primary inputs and where no synchronization elements and clock signal exist. In contrast, in a *sequential circuit*, a sequence of data, synchronized by a clock signal, is processed. A sequential circuit may contain a feedback loop through registers, such that the output at any given time is not only a function of the primary input but of previous outputs as well.

In this work, we model a sequential circuit by the schematic depicted in Figure 1. Note that this structure offers a unique representation of any given logical circuit without combinational loops that possibly contains multiple register stages. More precisely, every given circuit (without a combinational loop) can be represented as a *sequential circuit* that follows the structure shown in Figure 1, where all synchronization elements are packed into the main register stage, the combinational circuit receives the primary input and the outputs of the main register while the primary output is taken from the combinational circuit. As a side note, this illustrates a Mealy machine, which covers Moore machines as well [Mea55].

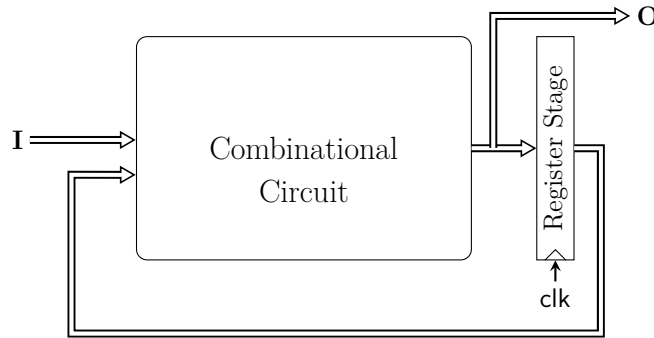
As it is explained in detail later in Section 3, we model the given unprotected circuit by the structure shown in Figure 1, hence extracting and processing the combinational part and the register stage individually to build the masked counterparts.

# 3 Technique

In this section, we gradually present the technical details of the procedure which AGEMA follows to generate a secure masked implementation from the given unprotected implementation. To this end, we first review the masking schemes which are currently supported by AGEMA.

## 3.1 PINI Hardware Gadgets

As PINI offers trivial composition of hardware gadgets in the robust probing model, we restrict our examples to known gadgets fulfilling this security notion. However, we would



**Figure 1:** General schematic of a sequential circuit.

like to stress that AGEMA offers a generic framework to dynamically substitute circuit parts with masked gadgets and is not restricted to any specific type of gadgets. Recently, several gadgets have been proposed that fulfill the PINI notion in the robust probing model. As they differ with respect to the logic function they realize, the fresh randomness they require and their latency, we describe and compare them in the following, where the number of required fresh randomness is denoted by  $r$  and the number of added register stages (i.e., the latency) by  $l$ .

### 3.1.1 HPC1

HPCs – proposed by Cassiers et al. in [CGLS21] – realize 2-input AND gadgets composable under the PINI notion in the robust probing model and are generic for arbitrary security orders. The authors introduced HPC1, which simply consists of a DOM-AND where the sharing of one input is refreshed. Hence, the added number of register stages is  $l = 2$ . The DOM-AND needs  $d(d + 1)/2$  bits of fresh masks for any given security order  $d$ . However, since the mask refreshing is expected to be SNI, the required number of additional fresh masks is identified through the table [1, 2, 4, 5, 7, 9, 11, 12, 15, 17] for security order  $d \leq 10$ .

### 3.1.2 HPC2

Cassiers et al. further proposed another construction for an AND gadget, HPC2, requiring  $r = d(d + 1)/2$  fresh randomness and  $l = 2$  added register stages for any security order  $d$ .

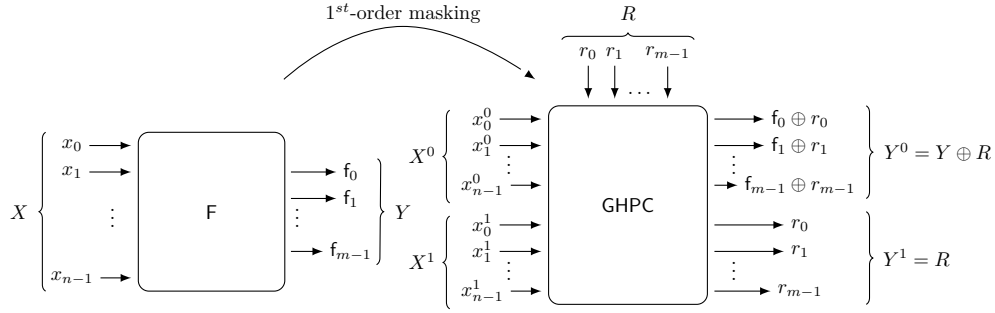
### 3.1.3 GHPC

Generic Hardware Private Circuits (GHPCs), introduced in [KSM22], allow the construction of gadgets realizing any (vectorial) Boolean function but are limited to first-order security. Here,  $l = 2$  is the number of added register stages, and the required number of fresh masks is  $r = 1$  per output bit, regardless of the Boolean function the gadget realizes.

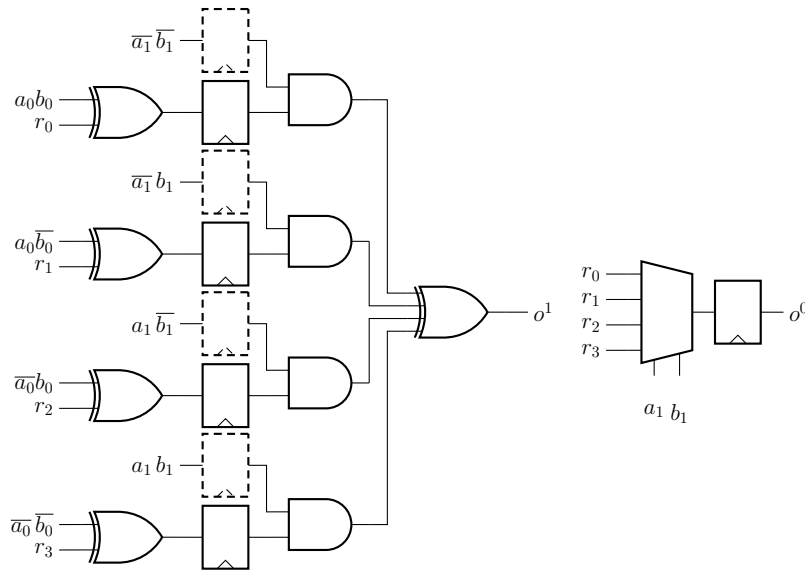
The concept of GHPC is depicted in Figure 2. Every input is split into two shares while the result of the gadget is simply the coordinate function  $f_i$  blinded by fresh randomness  $r_i$  for every  $0 \leq i < m$ , with  $m$  being the function’s output bit size.

### 3.1.4 GHPC<sub>LL</sub>

In [KSM22], the authors further introduced GHPC<sub>LL</sub>, a low-latency variant of GHPC, which requires only one register stage ( $l = 1$ ) to compute any vectorial Boolean function but needs  $2^n$  fresh random bits per output bit for a Boolean function with  $n$  inputs, i.e., in total  $m \times 2^n$  bits.



**Figure 2:** The GHPC concept of transforming any vectorial Boolean function into a first-order secure and composable gadget.



**Figure 3:** First-order GHPC<sub>LL</sub>-AND realizing  $o = ab$ . Dashed registers are optional but necessary to make a pipelined design.

A simple 2-input AND gadget realized as a GHPC<sub>LL</sub> is shown in Figure 3. This is the only known composable AND gadget with a latency of a single clock cycle.

### 3.1.5 HPC-MUX

Having an HPC-AND gadget, other 2-input non-linear gates can be easily constructed. By inverting one output share, the HPC-NAND is built, and by additionally inverting one share of both inputs the HPC-OR is constructed as  $\overline{a} \wedge \overline{b} = a \vee b$ . HPC-NOR can also be made this way. Note that XOR is trivial under PINI notion. In other words, if all non-linear sub-circuits fulfill the PINI requirements, XOR can be realized with no fresh randomness and no additional register stage by XORing the corresponding shares of the given inputs. XNOR can also be made by XOR while inverting one output share.

2-input multiplexers (MUXes) are commonly used in digital circuits as a building block (available in almost every standard logic library). Designing an efficient shared version of a 2-input MUX offering security under the PINI notion is straightforward as it can be directly derived using a single HPC-AND gadget. The Boolean function  $f$ , describing a MUX, where one of two inputs  $a \in \mathbb{F}_2$ ,  $b \in \mathbb{F}_2$  is selected by  $s \in \mathbb{F}_2$  can be rewritten as

$f = sa \oplus \bar{s}b = s(a \oplus b) \oplus b$ . Realizing a composable, shared multiplexer for arbitrary security orders is hence possible by means of two trivial XOR operations and a single HPC-AND gadget. As a result, the randomness requirements and the latency is inherited from the instantiated HPC-AND gadget, i.e., whether an HPC1-, HPC2-, GHPC-, or GHPC<sub>LL</sub>-AND is employed, where the two latter cases are restricted to constructing first-order secure designs only.

### 3.2 Procedure

Since the main goal is the conversion of an unprotected implementation to a masked one, we first have to analyze the netlist of the unprotected implementation. In other words, the unprotected (behavioral) implementation should be first synthesized by a synthesizer, e.g., Design Compiler [Inc] or Yosys [Wol]. The resulting Verilog netlist<sup>2</sup> is then given to AGEMA. Note that AGEMA has a custom library file, where the designer should specify the details of each cell, e.g., their input and output ports. Therefore, the synthesizer should also be set to just use a restricted list of cells to generate the Verilog netlist, typically only NOT, 2-input AND, NAND, OR, NOR, XOR, XNOR, MUX, and D flip-flops.

Then, as a first step, AGEMA builds a graph based on the given Verilog netlist, and represents the circuit following the concept given in Section 2.6 as shown in Figure 1, i.e., a combinational circuit and a single main register stage. Naturally, not necessarily all parts of the given design should be masked, e.g., excluding the control logic. Further, the designer may desire to not mask the key, for example if protection against profiling attacks targeting the key schedule is excluded (for such cases, see [MS16, PMK<sup>+</sup>11, UHA17, SM20], where no key masking is applied). In tweakable block ciphers, e.g., CRAFT [BLMR19], the tweak is supposed to be public knowledge. Hence, in a corresponding encryption/decryption implementation, there is no need to mask the tweak. In AGEMA, this is supported by setting the attribute of the primary input signals. If a signal is annotated as *secure*, it will be converted to a masked form with  $d + 1$  shares in the resulting masked circuit, while  $d$  (the order of masking) is also defined by the designer.

Hence, the next step of the process is to identify which parts of the given circuit should be masked based on the attribute of the primary inputs defined by the designer. If any input of a cell is marked secure, its output should also be marked secure. Therefore, we propagate the secure signals through all cells of the circuit until no new signal is marked as secure. Note that this includes the main registers and their role as the input to the combinational circuit.

In order to formalize this process, we model the given logic circuit as a Directed Acyclic Graph (DAG)  $G = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{Y})$  with nodes in  $\mathcal{V}$  and edges in  $\mathcal{E}$ . More precisely, any Boolean function  $\mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ , over inputs  $\mathcal{X} = \{x_i | 1 \leq i \leq n\}$  and outputs  $\mathcal{Y} = \{y_i | 1 \leq i \leq m\}$ , can be modeled as DAG which is defined as follows.

**Definition 2** (Syntax of DAG). Given a finite DAG  $G = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{Y})$  with vertices  $\mathcal{V}$ , edges  $\mathcal{E}$ , primary inputs  $\mathcal{X}$ , and primary outputs  $\mathcal{Y}$ , an  $m$ -rooted DAG is defined as follows:

- (1) Each vertex  $v \in \mathcal{V}$  represents a single combinational or sequential gate in the netlist.
- (2) Each edge  $e \in \mathcal{E}$  represents a single wire carrying an element from the finite field  $\mathbb{F}_2$ .
- (3) There are exactly  $n$  terminal nodes  $v$ , each labeled with a unique  $x_i \in \mathcal{X}$ .
- (4) There are exactly  $m$  root nodes  $v$ , each labeled with a unique  $y_i \in \mathcal{Y}$ .

Based on this, Algorithm 1 shows the procedure to propagate the secure signals into the circuit and how we split the circuit into two parts: the *secure zone* and the *normal zone*.

<sup>2</sup>This can be set in a script executed by the synthesizer to generate a Verilog netlist as the result of the synthesis.

**Algorithm 1** Propagation of secure signals

---

<b>Input:</b> $G : \{\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{Y}\}$	▷ given circuit
<b>Input:</b> $\mathcal{X}_s : \{x \in \mathcal{X} \mid \text{attribute}(x) = \text{secure}\}$	▷ secure primary inputs
<b>Output:</b> $\{\mathcal{V}_s, \mathcal{E}_s, \mathcal{X}_s, \mathcal{Y}_s\}$	▷ <i>secure zone</i>
<b>Output:</b> $\{\mathcal{V}_n, \mathcal{E}_n, \mathcal{X}_n, \mathcal{Y}_n\}$	▷ <i>normal zone</i>

---

```

1:  $\mathcal{V}_s \leftarrow \emptyset, \mathcal{E}_s \leftarrow \emptyset$ 
2: for  $\forall x \in \mathcal{X}_s$  do
3:    $\mathcal{E}_s \leftarrow \mathcal{E}_s \cup \text{output}(x)$ 
4: end for

5: repeat
6:    $\mathcal{V}'_s \leftarrow \emptyset$ 
7:   for  $\forall v \in \mathcal{V} \setminus \mathcal{V}_s$  do
8:     if  $\exists \text{input}(v) \in \mathcal{E}_s$  then
9:        $\mathcal{V}'_s \leftarrow \mathcal{V}'_s \cup v$ 
10:       $\mathcal{E}_s \leftarrow \mathcal{E}_s \cup \text{output}(v)$ 
11:     end if
12:   end for
13:    $\mathcal{V}_s \leftarrow \mathcal{V}_s \cup \mathcal{V}'_s$ 
14: until  $\mathcal{V}'_s \neq \emptyset$ 
15:  $\mathcal{Y}_s \leftarrow \mathcal{Y} \cap \mathcal{V}_s$ 

16:  $\mathcal{V}_n \leftarrow \mathcal{V} \setminus \mathcal{V}_s, \mathcal{E}_n \leftarrow \mathcal{E} \setminus \mathcal{E}_s, \mathcal{X}_n \leftarrow \mathcal{X} \setminus \mathcal{X}_s, \mathcal{Y}_n \leftarrow \mathcal{Y} \setminus \mathcal{Y}_s$ 

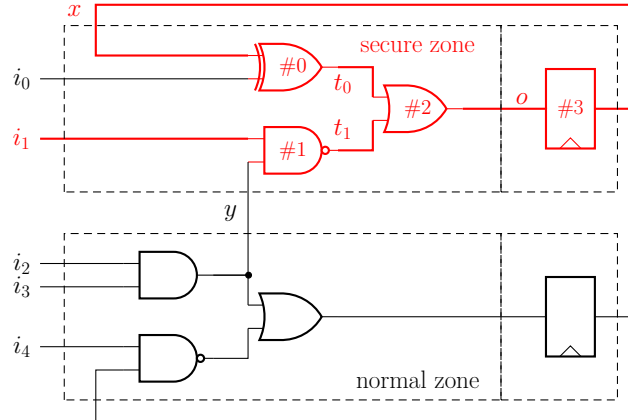
```

---

Figure 4 shows a simple example after the application of this algorithm, where the signals and cells which should be masked are highlighted in red. In this example,  $i_1$  is the only primary input which is marked as *secure* by the designer, i.e., this primary input should be masked ( $\mathcal{X}_s = \{i_1\}$ ). Then, the algorithm adds cell #1 and its output  $t_1$  to the secure zone. Next, cell #2 and its output  $o$  are added, and then the register cell #3 and its output  $x$ . In the next round, cell #0 and its output  $t_0$  are marked as secure, and the algorithm terminates since no other cells can be added to the secure zone. At the end, as shown in Figure 4, by selecting just  $i_1$  as the secure primary input, cells #0 to #3 and signals  $t_i, t_0, t_1, o$ , and  $x$  are identified to be masked. Note that other cells and intermediate signals including  $y$  and the primary input  $i_0$  (which are inputs to the secure zone) are not masked. We deal with such a combination in Section 3.3.1. It is noteworthy that we re-use this exemplary circuit in the next sections to illustrate different processing methods.

### 3.3 Processing Methods

The next step is to construct the masked variant of the secure zone. Apart from the fact that different masking schemes are supported (see Section 3.1), we can process the secure zone and build a more optimized netlist in favor of the selected masking scheme. To this end, AGEMA supports four different processing methods explained below with an example for each method in Figure 5 which is based on the secure zone identified in Figure 4. Such different processing methods – explained in the following – do not necessary require the Verilog netlist of the given unprotected circuit. However, in order to keep a constant form for the input of AGEMA, we developed the processing algorithms to just deal with such a representation of the given circuit. As a side note, we like to mention that all processing methods can be freely combined with all supported masking schemes, except ANF which is dedicated to GHPC and GHPC<sub>LL</sub>. An overview of the possible combinations is given in Table 1.



**Figure 4:** Exemplary circuit after the propagation of secure signals.  $i_1$  is the only primary input marked as secure. The red signals and cells indicate the extracted parts which should be masked.

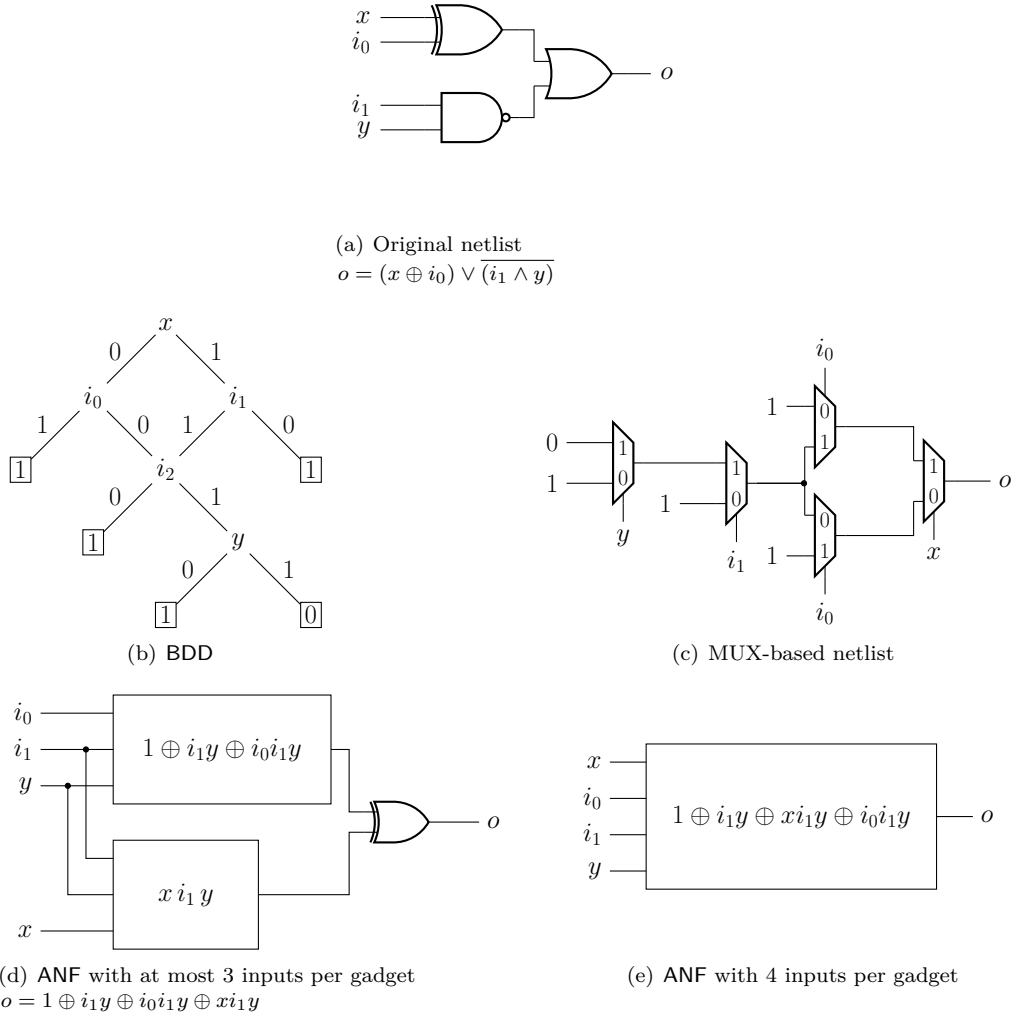
**Table 1:** Supported masking schemes and processing methods.

Processing Method	Masking Scheme		
	HPC1/HPC2 ( $d \geq 1$ )	GHPC ( $d = 1$ )	GHPC <sub>LL</sub> ( $d = 1$ )
Naive	✓	✓	✓
BDD <sub>SYLVAN</sub>	✓	✓	✓
BDD <sub>CUDD</sub>	✓	✓	✓
ANF		✓	✓

### 3.3.1 Naive

Every cell in the netlist of the secure zone can be naturally exchanged with its masked variant depending on the selected masking scheme. This also necessitates to replace every signal of the secure zone with its masked form with  $d + 1$  shares. Since this is just a translation of one netlist into another while keeping the original structure (i.e., the number of cells and how they are connected), the efficiency of the resulting masked circuit depends on how the original circuit has been synthesized. For example, the non-linear gadgets need fresh randomness and introduce register stages into the gates (see Section 3.1). Therefore, the number of non-linear gates and how they are composed (i.e., the logical depth of the circuit) has a direct effect on the number of required fresh masks and latency overhead of the resulting masked circuit.

We should also highlight that every signal which is not marked as secure but is involved in the secure zone is padded with 0 to form  $d + 1$  shares. Examples include the primary input  $i_0$  and the internal signal  $y$  in the example shown in Figure 4. Note that it should be carefully evaluated whether this may pose any security issue in the used gadgets. For example, cell #1 (NAND gate) in Figure 4 receives the masked  $i_1$  and unmasked  $y$  (i.e.,  $y$  padded with 0). Depending on the used gadget, this does not necessarily lead to a secure design. However, we have examined this in all our employed gadgets HPC1, HPC2, GHPC, and GHPC<sub>LL</sub>, and verified that this does not degrade their security.



**Figure 5:** Different processing methods for the combinational port of the secure zone of the exemplary circuit in Figure 4.

### 3.3.2 BDD

In discrete mathematics and computer science, Binary Decision Diagrams (BDDs) are often used as basic data structures to represent and manipulate Boolean functions. The seminal concept of BDDs has been introduced by Akers [Ake78] and refined by Bryant [Bry86], improving efficiency and conciseness through variable ordering. Nowadays, many applications in logic synthesis and formal verification of digital Integrated Circuits (ICs) rely on (reduced and ordered) BDDs<sup>3</sup>. This also holds for SILVER [KSM20] introduced in Section 2.5.

**Representation.** Multi-root BDDs provide a unique, concise, and canonical representation of Boolean functions  $\mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ . In particular, any multi-root BDD can be represented as a DAG with  $m$  root nodes and two terminal nodes  $\{\mathbf{0}, \mathbf{1}\}$ . More precisely, BDDs can be defined syntactically and semantically as follows.

**Definition 3** (Syntax of BDDs). Given a pair  $(\pi, \mathcal{G})$ , where  $\pi$  denotes a variable ordering

<sup>3</sup>For the sake of simplicity, we refer to Reduced Ordered Binary Decision Diagrams as BDDs.

and  $G = (\mathcal{V}, \mathcal{E})$  is a finite DAG with vertices  $\mathcal{V}$  and edges  $\mathcal{E}$ , the syntax of an  $m$ -rooted Reduced Ordered Binary Decision Diagram is defined as follows:

- (1) There are exactly  $m$  root nodes and each node  $v \in \mathcal{V}$  is either a non-terminal or one of the two terminal nodes  $\{\mathbf{0}, \mathbf{1}\}$ .
- (2) Each non-terminal node  $v \in \mathcal{V}$  is labeled with a variable, denoted as  $\text{var}(v)$  and has exactly two distinct child nodes in  $\mathcal{V}$ , which are denoted as  $\text{then}(v)$  and  $\text{else}(v)$ . More precisely, there is no non-terminal node  $v$  such that  $\text{then}(v) = \text{else}(v)$ .
- (3) There are no duplicate nodes, i.e., for each pair of nodes  $\{v, v'\} \in \mathcal{V}^2$  at least one of the following conditions holds:
  - (i) The variable label is different, i.e.,  $\text{var}(v) \neq \text{var}(v')$ .
  - (ii) The child nodes are different, i.e.,  $\text{then}(v) \neq \text{then}(v')$  or  $\text{else}(v) \neq \text{else}(v')$ .
- (4) For each path from root nodes to terminal nodes, the variable labels are encountered at most once and in the same order, defined by the variable ordering  $\pi$ .

Using the principle of *Shannon decompositions*, each multi-rooted BDD recursively defines a Boolean function  $\mathbb{F}_2^n \mapsto \mathbb{F}_2^m$  and arbitrary Boolean operations.

**Definition 4** (Semantic of BDDs). The representation of a Boolean function  $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ , defined over the input variables  $\mathcal{X} = \{x_i | 1 \leq i \leq n\}$ , is defined recursively according to the following specification:

- (1) Given a terminal node  $v$ , then  $f_v|_x = 0$  if  $v$  is the terminal node  $\mathbf{0}$ , and  $f_v|_x = 1$  otherwise.
- (2) Given a non-terminal node  $v$  and  $\text{var}(v) = x_i$ , then  $f_v$  is defined recursively by the Shannon decomposition:  $f_v = x_i \cdot f_{\text{then}(v)}|_{x_i=1} + \bar{x}_i \cdot f_{\text{else}(v)}|_{x_i=0}$ .
- (3) Given two root nodes  $v_1$  and  $v_2$  and any binary Boolean operation  $\circ$ , such that  $f = f_{v_1} \circ f_{v_2}$ , then  $f$  can be derived recursively as:

$$\begin{aligned} f &= x_i \cdot f|_{x_i=1} + \bar{x}_i \cdot f|_{x_i=0} \\ &= x_i \cdot (f_{v_1} \circ f_{v_2})|_{x_i=1} + \bar{x}_i \cdot (f_{v_1} \circ f_{v_2})|_{x_i=0} \\ &= x_i \cdot (f_{v_1}|_{x_i=1} \circ f_{v_2}|_{x_i=1}) + \bar{x}_i \cdot (f_{v_1}|_{x_i=0} \circ f_{v_2}|_{x_i=0}) \end{aligned}$$

**Transformation.** The purpose of the BDD processing method is to create a BDD for the secure zone netlist. More precisely, the Boolean function of the secure zone netlist is transformed into a multi-root BDD, whereas each node in the BDD corresponds to a MUX. Note, however, that in the context of BDDs, each select signal is connected to a primary input, while the data signals are connected to the constants  $\mathbf{0}$  or  $\mathbf{1}$  or any other MUX corresponding to a BDD node. Therefore, AGEMA extracts an equivalent netlist of the secure zone, purely based on 2-input MUXes, which afterwards are exchanged and replaced by their masked counterpart (see an example in Figure 5(b) and Figure 5(c)).

AGEMA employs two different C/C++ libraries for the construction and manipulation of BDDs, explained as follows.

**SYLVAN**<sup>4</sup> is a state-of-the-art BDD high-performance, multi-core decision diagram package implemented in C/C++. In particular, manipulation and processing of BDDs and binary operations has been extensively optimized and implemented for multi-core support, outperforming existing, but single-core BDD packages.

<sup>4</sup><https://github.com/utwente-fmt/sylvan.git>



**CUDD**<sup>5</sup> (Colorado University Decision Diagram) is a package for manipulation and processing of BDDs, Algebraic Decision Diagrams (ADDs), and Zero-suppressed Binary Decision Diagrams (ZDDs) implemented in C. In contrast to SYLVAN, CUDD provides an extensive set of features and operations that can be performed on BDDs, including automatic and dynamic reordering of the variables. Hence, although CUDD has been mostly designed for single-core processors, it can outperform SYLVAN in certain applications, mostly due to reduced memory requirements and BDD sizes (due to more optimal variable orderings).

**Limitations.** In contrast to the Naive processing method, the BDD transformation results in a unique representation (under a given variable ordering). As a result, the BDD representation is independent of the original netlist representation but solely depends on the underlying Boolean function, hence reducing the effort of optimizing the original and unprotected design. Further, since each BDD can be represented as a multiplexer cascade in digital logic, creation and optimization of a single masked MUX-gadget is sufficient to convert unprotected designs into protected designs (see Section 3.1.5). However, in contrast to common approaches, the primary inputs of the secure zone serve as selection signals of the multiplexers (instead of being connected to the multiplexer data inputs). As a consequence, the logical depth of the multiplexer cascade is solely limited by the number of primary inputs of the secure zone, hence, determining the resulting latency of the masked circuit.

Besides, since BDDs are only canonical under a given variable ordering, we employ two different state-of-the-art BDD libraries. While SYLVAN is a high-performance library captivating through multi-core algorithms and operations, in particular with respect to BDD generation and recombination, CUDD also supports automated and dynamic variable re-ordering. In fact, using some pre-defined and global thresholds, the library automatically performs variable re-orderings once the thresholds are exceeded, in order to find better (i.e., smaller) BDD representations through changing the ordering of the variables (i.e., primary inputs). As a smaller BDD directly translates to smaller masked circuits using fewer multiplexers, we decided to support and evaluate both BDD libraries for their various benefits and limitations.

### 3.3.3 ANF

As stated in Section 3.1, GHPC and its low-latency variant GHPC<sub>LL</sub> allow to construct first-order secure composable gadgets from arbitrary functions. In other words, in addition to GHPC and GHPC<sub>LL</sub> gadgets for 2-input non-linear gates (AND, NAND, OR, NOR) which can be used in other processing methods, GHPC and GHPC<sub>LL</sub> gadgets can directly be made for larger Boolean functions with arbitrary number of variables. However, with increasing variable dependencies, area overhead of the gadgets gradually becomes more obstructive. To this end, the ANF processing method tries to find trade-offs between single gadget size and overall circuit size.

**Representation.** In general, any Boolean function can be expressed canonically using several normal forms, such as Conjunctive Normal Form (CNF), Disjunctive Normal Form (DNF), or Algebraic Normal Form (ANF). In particular, the ANF representation is often considered for masking purposes due to the trivial masking of XOR operations.

**Definition 5** (Algebraic Normal Form). For any Boolean function  $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$  there exists a unique AND-XOR representation, called the ANF of  $f$ :

$$f(\mathbf{x}) = \bigoplus_{u \in \mathbb{F}_2^n} a_u \prod_{u_i=1} x_i \text{ with } a_u \in \mathbb{F}_2,$$

<sup>5</sup><https://github.com/ivmai/cudd.git>

where the summands of  $f$  are called *monomials*. Each monomial forms a conjunction of a unique subset of  $\mathbf{x}$  defined by its corresponding index  $u$ . The *degree* of a monomial is defined as the size of its input set. Furthermore, the *algebraic degree* of  $f$  is defined as the highest degree of all function's monomials.

**Transformation.** We first construct the ANF of the secure zone. The construction mechanism represents each gate (e.g., AND, OR, XOR) of the original netlist with its corresponding ANF. More precisely, our tool computes the ANF of any gate based on the primary inputs of the secure zone. To this end, the inputs of each gate are expressed in terms of an ANF given in the primary inputs.

However, for the construction of gadgets, we are only interested in the ANF of the outputs of the entire combinational part of the secure zone. Unfortunately, as stated above, constructing an individual gadget per secure zone output may lead to a very large circuit if the corresponding ANF depends on a very large number of primary inputs. As a consequence, to reduce the circuit size, we apply two optimization techniques introduced in the following. For the sake of simplicity and better understanding, we analyze an exemplary Boolean function  $f : \mathbb{F}_2^4 \mapsto \mathbb{F}_2$  represented by the following ANF.

$$f = x_0x_2 \oplus x_0x_3 \oplus x_0x_2x_3 \oplus x_1x_2 \oplus x_1x_3 \oplus x_1x_2x_3 \quad (1)$$

Now, the trivial approach is to construct the entire function as a single gadget. As  $f$  depends on four different input values, the corresponding gadget also requires four inputs. Nevertheless,  $f$  can be rewritten in a way that the linear combination  $x_0 \oplus x_1$  is processed instead of  $x_0$  and  $x_1$ , i.e.,

$$f = (x_0 \oplus x_1)(x_2 \oplus x_3 \oplus x_2x_3). \quad (2)$$

Note that finding suited linear combinations is not trivial. We explain our methodology in the following. Providing the linear combination  $(x_0 \oplus x_1)$  to the gadget instead of  $x_0$  and  $x_1$  reduces the input size of the gadget by one. In practice, such linear combinations occur in many modern block ciphers including a key addition operation. Hence, detection of such operations and extraction of linear combinations often allows to typically halve the number of inputs per gadget. However, the minimization of gadgets due to finding linear combinations is not only restricted to the gadgets inputs. For instance, considering  $h : \mathbb{F}_2^4 \mapsto \mathbb{F}_2$  as

$$h = x_0 \oplus x_1 \oplus x_0x_1 \oplus x_2 \oplus x_3 \oplus x_2x_3,$$

and computing the entire function in a single gadget is inefficient since the algebraic degree of  $h$  is smaller than the number of inputs. In particular, only  $x_0$  and  $x_1$  as well as  $x_2$  and  $x_3$  are combined non-linearly. In addition, only two different monomials of degree two ( $x_0x_1$  and  $x_2x_3$ ) exist, i.e., not all inputs are combined in conjunctions. Hence, splitting  $h$  into two functions  $h = h_0 \oplus h_1$  such that  $h_0 = x_0 \oplus x_1 \oplus x_0x_1$  and  $h_1 = x_2 \oplus x_3 \oplus x_2x_3$  results in two gadgets with only two inputs each. Compared to a single gadget with four inputs, the area footprint can be reduced. We also refer to Figure 5(d) and Figure 5(e), where two ANF gadgets for the exemplary circuit of Figure 4 are shown. As a result, linear output combinations, as described here, exist in many modern block ciphers where linear diffusion layers permute the outputs of multiple non-linear S-boxes which typically operate on a small set of inputs (e.g., 4 or 8 bits).

Now, given an arbitrary Boolean function representing a secure zone output in ANF, similar to the examples shown above, we can split the entire ANF into multiple sub-functions of independent inputs. For instance, considering a full round of a block cipher, this step is beneficial as different S-boxes are usually computed on non-colliding sets of input variables. Therefore, we should be able to construct gadgets that operate only on a small set of input values (depending on the S-box input size). More precisely, in order to

find suitable sub-functions, we first extract all monomials with maximal algebraic degree and place them into sub-functions. Specifically, all monomials that share no input are placed in different sub-functions, while monomials sharing at least one input are placed in the same sub-function. In the next step, we extract all smaller monomials of the ANF and place them in one of the existing sub-functions such that each monomial is placed in a sub-function if it shares inputs with the largest monomial of this sub-function. Eventually, we repeat this procedure for every output ANF while adding new outputs to the gadgets. Hence, each gadget may be used to compute sub-functions of multiple output ANFs if they depend on the same inputs. As a result, each output ANF can be computed as the sum of different sub-functions while each sub-function receives a different set of inputs. At this point, the gadgets are independent of each other and we can optimize them individually.

For some lightweight block ciphers, such as PRINCE [BCG<sup>+</sup>12], PRESENT [BKL<sup>+</sup>07], Midori [BBI<sup>+</sup>15], Skinny [BJK<sup>+</sup>16], and CRAFT [BLMR19], the diffusion layer only combines the output of different S-boxes. In other words, the diffusion layer never combines different output bits of the same S-box. As the gadgets will be optimized at most up to the S-box sizes, the sub-functions and the outputs of the S-boxes become equivalent. Prominent exceptions of this rule are the AES [DR02] and LED [GPPR11], where the MixColumns also linearly combines the outputs computed by the same S-box (through Galois-field multiplication with constants in the MixColumns matrix). Note that since we are analyzing the netlist of the secure zone, such a linear combination is not trivially visible in the output ANFs. This translates to gadgets with a large number of outputs as the gadgets compute all linear combined outputs separately and not the small set of S-box outputs resulting in a high area and fresh-randomness overhead.

In order to detect such linear combinations and reduce the number of gadget outputs (ideally) up to the number of S-box outputs, we search for a minimal set of different functions whose linear combinations compute all required sub-functions. We perform such a search with simulated annealing [KGV83], a discrete optimization technique that searches for a minimal solution by evaluating solutions that are similar to the current solution (so-called neighbors). The advantages of simulated annealing compared to other optimization techniques such as constraint programming [Apt03] are the great performance and the ability to escape local minima. This is achieved since the acceptance of a neighbor is partially probabilistic. Hence, sometimes also bad neighbors are accepted to escape local minima. We start with an input solution  $x$  (a set of functions) that computes all sub-functions separately. Therefore, each sub-function is given as a single output and with a single summand, what translates to a list with a single element. During the simulated annealing, we split our initial solution into multiple summands that compute the outputs with a minimal number of different summands. We show our method in Algorithm 2. For the neighboring function  $\text{neighbor}(z)$ , we randomly modify input solution  $z$  by selecting one summand from a random sub-function (at the beginning, each function is given as a single summand), XOR it to every occurrence of another randomly-chosen summand, and insert it to every modified sub-function. Moreover, we define our objective costs  $\text{cost}(z)$  of the solution  $z$  as the number of different summands required to compute all sub-functions. It turns out that a very small number of iterations and a very low cooling factor are enough to recover the minimal set of gadget outputs. In our experiments, we used a cooling factor  $c$  of 0.9 and start with  $n = 100$  iterations per cooling step which increases by 100 after every cooling step. The initial temperature is  $t = 1$  and cooled down until it reaches  $t' = 0.5$ . Note that the set of gadget outputs not necessarily encompasses the outputs of the underlying S-boxes but linear combinations which also result in a minimal number of gadget outputs.

After optimizing the gadgets outputs we investigate each gadgets inputs. As shown above in Equation (2), a gadget can depend on a linear combination of its inputs. In order to detect such cases for a given gadget, for each input (e.g.,  $x_0$ ) we first make a set  $\mathcal{L}_{f,x_0}$  of

**Algorithm 2** Search for optimal gadget outputs

---

**Input:**  $t, t', c, n, x$   $\triangleright$  Simulated annealing parameters: Temperature  $t$ , minimum temperature  $t'$ , cooling factor  $c$ , iterations per cooling step  $n$ , initial solution  $x$

**Output:**  $z$   $\triangleright$  A local-optimum solution

```

1:  $z \leftarrow x$ 
2: while  $t > t'$  do
3:   for  $i = 1, \dots, n$  do
4:      $y \leftarrow \text{neighbor}(z)$ 
5:      $\delta \leftarrow \text{cost}(y) - \text{cost}(z)$ 
6:      $r \xleftarrow{\$} [0, 1] \subset \mathbb{R}$   $\triangleright$  Random value in range  $[0,1]$ 
7:     if  $\delta \leq 0$  or  $e^{-\delta/t} > r$  then
8:        $z \leftarrow y$ 
9:     end if
10:  end for
11:   $t \leftarrow t \cdot c$ 
12: end while

```

---

**Algorithm 3** Search for linear input combinations

---

**Input:**  $\mathcal{L}_{G_i}, \mathcal{L}_x$   $\triangleright$  List of the gadgets output functions and inputs

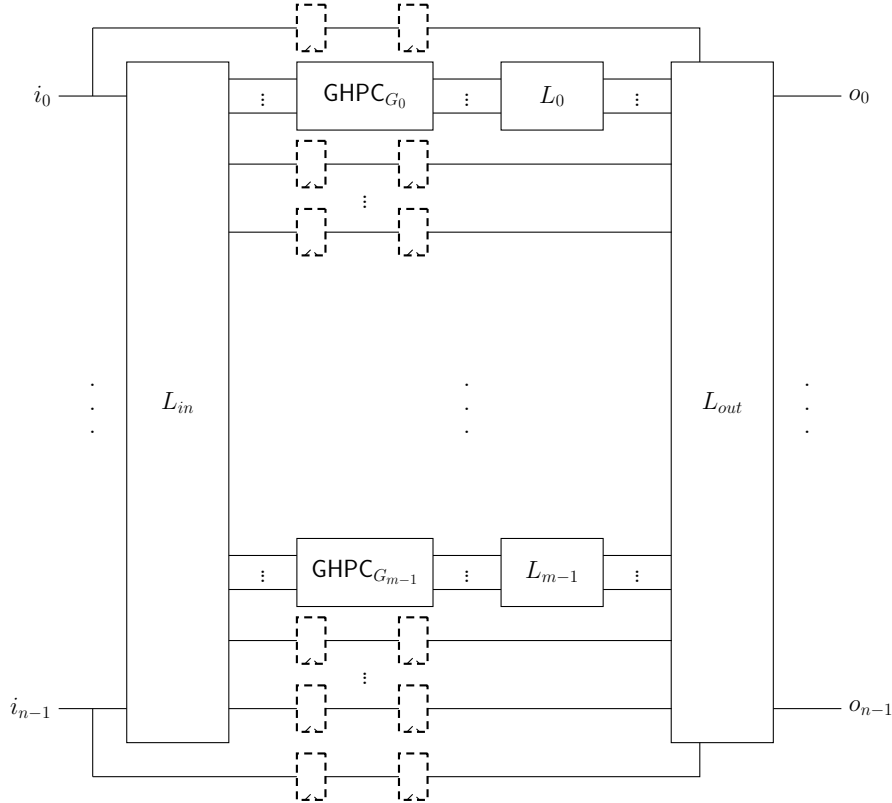
**Output:**  $\mathcal{L}_{G_o}$   $\triangleright$  List of the gadgets substituted output functions

```

1:  $\mathcal{L}_{G_o} \leftarrow \mathcal{L}_{G_i}$ 
2: for  $\forall (x_0, x_1) \in \mathcal{L}_x \times \mathcal{L}_x$  do
3:   if  $x_0 \neq x_1$  then  $\triangleright$  Get two different inputs of the gadget
4:     for  $\forall \mathcal{F} \in \mathcal{L}_{G_o}$  do  $\triangleright$  By  $\mathcal{F}$  we refer to all monomials of output function  $f$ 
5:        $\mathcal{L}_{f, x_0} \leftarrow \emptyset$ 
6:        $\mathcal{L}_{f, x_1} \leftarrow \emptyset$ 
7:       for  $\forall \mathcal{M} \in \mathcal{F}$  do  $\triangleright$  Get a monomial of the output function
8:         if  $x_0 \in \mathcal{M}$  then  $\mathcal{L}_{f, x_0} \leftarrow \mathcal{L}_{f, x_0} \cup \mathcal{M} \setminus \{x_0\}$ 
9:         end if
10:        if  $x_1 \in \mathcal{M}$  then  $\mathcal{L}_{f, x_1} \leftarrow \mathcal{L}_{f, x_1} \cup \mathcal{M} \setminus \{x_1\}$ 
11:        end if
12:      end for
13:    end for
14:    if  $\forall \mathcal{F} \in \mathcal{L}_{G_o}, \mathcal{L}_{f, x_0} = \mathcal{L}_{f, x_1}$  then
15:       $\mathcal{N} \leftarrow \{x_0, x_1\}$   $\triangleright$  Create the new linear combination
16:       $\mathcal{L}_x \leftarrow \mathcal{L}_x \setminus \{\{x_0\}, \{x_1\}\} \cup \mathcal{N}$   $\triangleright$  Update the gadgets inputs
17:      for  $\forall \mathcal{F} \in \mathcal{L}_{G_o}$  do
18:        for  $\forall \mathcal{M} \in \mathcal{F}$  do  $\triangleright$  Substitute the inputs with their linear combination
19:          if  $x_0 \in \mathcal{M}$  then  $\mathcal{M} \leftarrow \mathcal{M} \setminus \{x_0\} \cup \mathcal{N}$ 
20:          end if
21:          if  $x_1 \in \mathcal{M}$  then  $\mathcal{F} \leftarrow \mathcal{F} \setminus \mathcal{M}$ 
22:          end if
23:        end for
24:      end for
25:    end if
26:  end if
27: end for

```

---



**Figure 6:** Generic structure of a masked circuit after applying the ANF processing method. Dashed registers are optional but necessary to make a pipelined design.

all monomials that exist in output function  $f$  and include  $x_0$ . As only monomials including  $x_0$  are in  $\mathcal{L}_{f,x_0}$ , we erase  $x_0$  from the monomial before storing it. Then, we search for input combinations by iterating over all pairs of inputs, e.g.,  $(x_0, x_1)$ , and examining the corresponding sets  $\mathcal{L}_{f,x_0}$  and  $\mathcal{L}_{f,x_1}$ . Since we erase  $x_0$  from all monomials in  $\mathcal{L}_{f,x_0}$  and  $x_1$  from all monomials in  $\mathcal{L}_{f,x_1}$  both sets are exactly equivalent iff they differ only in  $(x_0, x_1)$ . Hence, equivalence shows that we can replace  $(x_0, x_1)$  by its linear combination. Naturally, two inputs  $x_i$  and  $x_j$ , such that  $i \neq j$ , can be replaced by their linear combination  $x_i \oplus x_j$  in an ANF, if both  $x_i$  and  $x_j$  are similarly combined with other inputs in all monomials. This is given if both sets of monomials are equal and non-empty for every output function of the gadget. If both conditions are met,  $x_i$  and  $x_j$  can be replaced with their linear combination  $x_i \oplus x_j$  in the entire gadget.

As a short example, we verify the linear combination in  $f$  given in Equation (1). For the input pair  $(x_0, x_1)$  it holds that  $\mathcal{L}_{f,x_0} = \{x_2, x_3, x_2x_3\}$  and  $\mathcal{L}_{f,x_1} = \{x_2, x_3, x_2x_3\}$ . Since it holds that  $\mathcal{L}_{f,x_0} = \mathcal{L}_{f,x_1}$ , the linear input combination  $(x_0, x_1)$  can be applied. This reduces the number of inputs of the gadget and the complexity of the computed function. We formalize our technique in Algorithm 3. Internally, we represent each monomial as a set of its inputs and each function as a set of monomials. Hence, we can represent a gadget ( $\mathcal{L}_{G_i}$  and  $\mathcal{L}_{G_o}$  in the algorithm) as a set of its output functions.

Finally, the result of the ANF processing method is a combination of gadgets and linear layers. A general structure is depicted in Figure 6. Initially, input layer  $L_{in}$  computes all linear input combinations which are fed to the gadgets computing all non-linear components. Different outputs of the  $i$ -th gadget are then linearly combined (through  $L_i$ ). Finally, the output layer  $L_{out}$  linearly combines different gadgets outputs.

**Limitations.** As already stated, although the other previously discussed processing methods can be combined with different masking schemes, ANF is purely dedicated to GHPC and GHPC<sub>LL</sub>. Hence, ANF can only generate first-order secure circuits. Similar to BDD, ANF generates a unique ANF of the outputs independent of the optimization level of the given netlist. Nevertheless, the result of ANF is not unique due to the probabilistic characteristic of simulated annealing. Hence, improvements in terms of area are possible by multiple executions of the tool while always selecting ANF as the processing method. In particular, the algorithms generate inefficient gadgets if the complexity of the secure zone grows. The given parameters for simulated annealing are suited for the optimization of typical S-boxes (up to 8-bit input and output sizes). Adjusting the parameters of simulated annealing could be helpful for more complex secure zones but can increase the runtime in return. Up to now, all gadgets are instantiated in parallel, leading to a fixed latency of two clock cycles for GHPC (resp. one cycle for GHPC<sub>LL</sub>). On the other hand, the largest gadget cannot be smaller than dictated by the highest algebraic degree of the output functions.

As stated before, BDD and ANF can be constructed from various representations of the given circuit. For the sake of having a unique form for the inputs to AGEMA, we only process Verilog netlist of the synthesized circuit.

### 3.4 Optimization

Up to this point, we have explained how the secure zone is extracted from the netlist and how it can be translated to a masked circuit. Depending on the chosen processing method and the masking scheme and more importantly the initial netlist of the secure zone, the resulting masked circuit introduces additional latency (more clock cycles) to the circuit and demands for a high or low number of fresh masks. Further, the performance of the resulting circuit is heavily affected by the multiplexers of the secure zone (if any). We have already given an efficient way to realize an HPC-MUX in Section 3.1.5. However, it is commonly seen that the secure zone contains multiplexers whose select signal is not marked as secure, i.e., not masked. For example, a plaintext which is given as the primary input is loaded under certain conditions, e.g., when the reset signal is high (or low). As another example, different computations are performed in different clock cycles, e.g., last round of the cipher is different to the other rounds (e.g., MixColumns is missing in the last round of AES), or in a serialized architecture during some clock cycles the output of the S-box is taken, and in some other clock cycles that of the diffusion layer. In such cases, there is no need to mask and translate the multiplexer with an HPC-MUX. Similar to the XOR, which is secure under PINI notion, an ordinary (unmasked) MUX can be straightforwardly instantiated  $d + 1$  times (for security order  $d$ ). Note that security under the PINI notion requires every signal to have an independent sharing [CS20]. Hence, connecting corresponding shares of two masked signal to an ordinary MUX controlled by an insecure signal would not violate any security requirements. This would greatly improve the efficiency of the resulting masked circuit. As a side note, the synthesizer should be directed to make use of MUXes in such cases. If the functionality of a MUX is realized by Boolean gates (AND, OR, XOR, etc.) and the netlist of the secure zone is optimized (e.g., for area, latency, or power through the synthesizer), it would not be straightforward to detect the MUXes in the secure zone, and most likely the resulting circuit would suffer from a high number of added register stages and a high demand for fresh randomness.

### 3.5 Synchronization

Since masked gadgets often have internal register stage(s), after applying the selected masking scheme, the combinational part of the secure zone is not fully combinational anymore. Therefore, the circuit would not necessarily work properly. Hence, the circuit

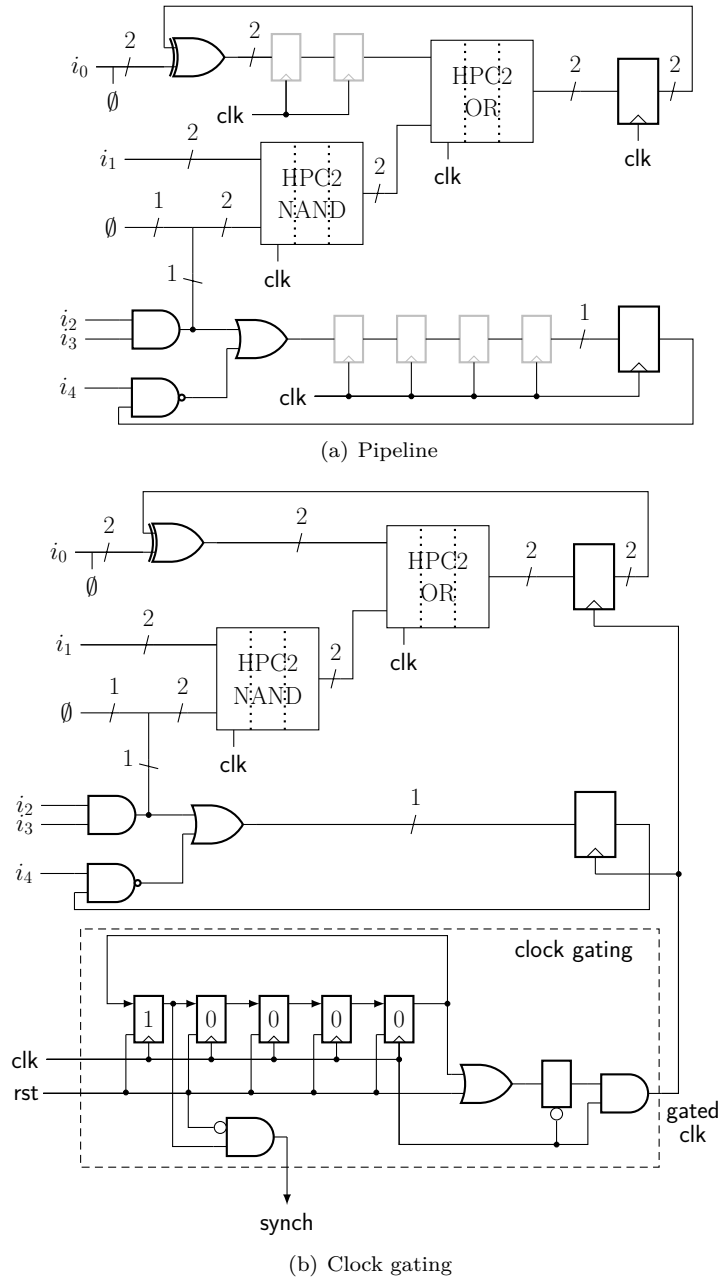
should be adjusted to keep its correct functionality. We achieve this by two different techniques explained below.

**Pipelining.** We can add extra registers to synchronize all inputs of every gadget as well as all inputs of the main register stage. An example is shown in Figure 7(a), which is based on the circuit depicted in Figure 4. Each HPC2 gadget introduces two register stages. Hence, in order to synchronize the inputs of the HPC2-OR gate in Figure 7(a) we need to place two cascaded registers at its first input (those which are marked by the gray color). This procedure is done by synchronizing all inputs of each gadget processed in order of their logic depth. At the end, all inputs of the main register stage are also synchronized. For the example shown in Figure 7(a), four registers are placed in the normal zone to synchronize it with the output of the secure zone. This way, the circuit keeps its correct functionality while realizing a pipeline design with  $p + 1$  stages if  $p$  register stages are added to the circuit (in the shown example,  $p = 4$  as two HPC2 gadgets are cascaded, each with 2 register stages). Hence, the circuit can process  $p + 1$  consecutive and independent inputs. We should highlight that the area overhead of the resulting circuit is relatively high, but it constructs a circuit with a high throughput due to its underlying pipeline architecture.

We should highlight that AGEMA is not able and does not try to detect control logic of the given circuit. As explained above, it just synchronizes all inputs of every gadget by instantiating register(s) at one of the inputs. The same is performed on the inputs of the main register stage in both secure and normal zones. This is adequate to build a fully-pipeline circuit with the same functionality as in the given unprotected circuit, but with  $p$  times higher latency. In other words, if the unprotected circuit has a latency of  $l$  clock cycles while processing a given input, the constructed pipeline masked counterpart has a latency of  $(p + 1)l$  clock cycles while consecutively processes  $p + 1$  given inputs. Note that this is dedicated to neither a certain implementation architecture nor a special handshaking fashion. This is a valid construction when the circuit is modeled in accordance with Section 2.6.

**Clock Gating.** In order to mitigate the area overhead of the former technique, we can make use of clock gating. More precisely, we need to make sure that the main register stage keeps its value until the computation of the secure zone is terminated. The same holds for the primary inputs. Hence, we do not add any extra registers to the circuit, but change the clock of the main register stage. This way, all internal registers of the gadgets are controlled by the main clock, whereas the main register stage is controlled by an added gated clock enabled once per evaluation of the secure zone. A circuit which is equivalent to the former example is shown in Figure 7(b). In order to keep the generality, a clock gating module is added to the design which can be adjusted based on the latency of the masked secure zone, i.e.,  $p$ . To this end, the clock gating module instantiates a rotating shift register with  $p + 1$  bits initialized by  $1\{0\}^p$  using an added control signal `rst`. Hence, every  $p + 1$  cycles the main register stage is clocked to proceed with the next round of the calculation of the secure zone. As a result, the latency of the clock-gated circuit is the same as the pipeline one, but it has a lower throughput as well as lower area overhead. More precisely, if the unprotected circuit has a latency of  $l$  clock cycles while processing a given input, the constructed clock-gating masked counterpart needs  $(p + 1)l$  clock cycles to accomplish the processing of a single given input.

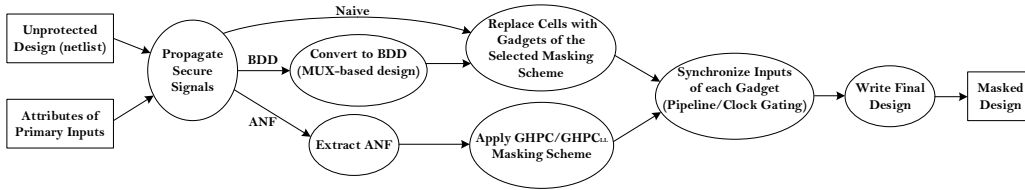
Similar to the pipelining, AGEMA does not analyze the circuit to detect any part related to control logic. We have developed a generic HDL code for the clock gating module with  $p$  as its input parameter. AGEMA just instantiates this module, adjusts  $p$ , and changes the clock of the main register stage to be provided by the clock gating module. This way, the circuit becomes equivalent to the given unprotected circuit, but with  $p$  times higher latency.



**Figure 7:** Different architectures: pipeline versus clock gating for the secure zone of the exemplary circuit in Figure 4 processed by the Naive method using HPC2 masking scheme.

Note that since the primary inputs are only allowed to change once per evaluation cycle right after the main register stage is clocked, the **clock gating** module generates an additional output signal **synch** to let the outer modules synchronize. More precisely, a positive edge is seen on the **synch** signal indicating that the primary inputs may change.





**Figure 8:** General procedure of AGEMA to generate a masked circuit.

### 3.6 Finalize

Based on the explanations given above, the procedure that AGEMA follows to generate a masked design can be summarized in Figure 8. At the end, based on the settings defined by the designer (i.e., selected processing method, masking scheme, security order  $d$ , pipeline or clock-gated design) AGEMA generates a new netlist for the masked design, where – in addition to the unmasked cells of the unprotected design – gadgets are instantiated. However, the RTL of the gadgets (which support different security orders and pipelining feature as well) should be provided separately, which are indeed global and the same for every design. We already provided the HPC1, HPC2, GHPC, and GHPC<sub>LL</sub> gadgets for the NOT, 2-input gates (AND, NAND, OR, NOR, XOR, XNOR), MUX and D flip-flop (can be found in the GitHub: <https://github.com/Chair-for-Security-Engineering/AGEMA>). If the designer wants to use any other gadget realizing any other gate, its specification should be defined in the AGEMA library (e.g., name and size of the ports, how many fresh masks are required, and how many cycles latency it has) and the RTL of the new corresponding gadget(s) should be provided. Hence, the generated netlist together with the RTL of the gadgets need to be synthesized for the target platform, e.g., any ASIC library or an FPGA.

## 4 Case Studies

In order to examine the efficiency and performance of circuits constructed by AGEMA, we evaluated several designs including different S-boxes and full cipher implementations under different settings, i.e., different security orders (up to 4th), various processing methods and different masking schemes.

### 4.1 S-boxes

We start with the 4-bit S-box of Skinny [BJK<sup>+</sup>16] and provide two different representations. In the first one, we straightforwardly implemented the S-box by a lookup table. The synthesizer translates such a behavior representation to a netlist, which is then given to AGEMA for further processing. For the second one, we followed the optimized representation provided in [CGLS21]. The corresponding results are given in Table 3 and Table 4 respectively. As the results are extensive and many tables are presented, all performance results are given in Appendix A. Note that all syntheses have been done using Synopsis Design Compiler and a NanGate 45 nm standard cell library. For these analyses, we covered all processing methods Naive, BDD<sub>SYLVAN</sub>, and BDD<sub>CUDD</sub> for masking scheme HPC2 at different security orders. For the sake of comparison, we covered HPC1 only for Naive method. ANF is also covered as a processing method where transformation into a secure design is only possible in combination with GHPC or GHPC<sub>LL</sub> (see Table 1). The effect of the given netlist on the performance of the masked circuit can be easily seen in the results associated to Naive processing method. The (HPC2, Naive)-approach for the lookup table based S-box adds 10 clock cycles to the latency compared to 4 clock cycles for the

optimized S-box. ANF and BDD methods are actually not affected by the optimality of the given netlist as they reconstruct the netlist. Further, it can be seen that HPC1 leads to a lower area overhead while it certainly demands for more fresh randomness.

We repeated this procedure for the AES S-box. In addition to a lookup table based representation, we took the Canright version [Can05] and the optimized design presented in [BP12], which – in addition to the linear layers (isomorphisms) – has at most 4 cascaded 2-input AND gates, making it suitable for a masked design. Performance results are given in Table 5, Table 6 and Table 7. The effect of the optimality of the given netlist on the performance (area and latency) is even clearer compared to the former case study.

## 4.2 Full Ciphers

For the full cipher implementations, we cover the list given in Table 2. The performance results are shown in Tables 10-14 in Appendix A. For all such case studies, we considered the following facts.

- For all designs, we marked the plaintext/ciphertext and the key as secure for AGEMA. In other words, the resulting masked circuit receives all inputs (except the control signals, e.g., `clk` and `rst`) in a masked form with  $d + 1$  shares and provides the output also with  $d + 1$ .
- If possible and available, we provided an optimized representation of the S-box. Above, we have given the source of such optimized designs for the Skinny and AES S-boxes. For PRESENT and LED, which share the same S-box, we took the optimized S-box representation from [CGLS21]. However, for Midori and CRAFT, which also share the same S-box, such representations are not available. Therefore, we represented the S-box by a lookup table. It can be seen in the performance results of CRAFT and Midori that the added latency (for Naive method) is higher compared to the other ciphers with an optimized 4-bit S-box.
- We hard-coded the multiplexers (controlled by Finite State Machine (FSM) or primary input control signals like `rst`) and directed the synthesizer to not optimize them (see Section 3.4). The same holds for XORs. If the XORs are also merged in other combinational circuits, the synthesizer may optimize in other directions, leading to a netlist with more (cascaded) non-linear gates.
- As explained in Section 3.3.2, BDD processing methods are not necessarily efficient for large combinational circuits when an optimized representation is available. This can be seen for Midori and CRAFT, where the S-box is based on a look-up table representation and BDD methods have the same latency overhead, while this does not hold true for the other ciphers, where an optimized representation of the S-box is given. Further, in AES round-based implementation, the round function, including 16 S-boxes followed by the MixColumns and 4 S-boxes of the KeySchedule, is too large to be processed by BDD methods.
- We reported two columns for the latency in Tables 10-14 in Appendix A. The “added latency” indicates the number of cycles which are added to each clock cycle of the unprotected implementation. The “full latency” is then calculated based on the added latency and the latency of the unprotected implementation. For example, the unprotected Skinny-64-64 round-based encryption needs 33 clock cycles to accomplish the encryption, and based on Table 10, the HPC2 Naive implementation adds 4 cycles latency. This results in the clock-gating implementation requiring  $33 \times (1 + 4) = 165$  clock cycles for an encryption. The pipeline implementation has the same latency, but processes  $4 + 1$  plaintexts consecutively in those 165 clock cycles. Hence, its throughput (ignoring the delay) stays the same as the unprotected implementation,

**Table 2:** Full cipher implementation case studies.

Cipher	Implementation	Reference
<b>AES-128</b>	Byte-serial and round-based encryption	[DR02]
<b>Skinny64</b>	Round-based encryption with 64-bit key	[BJK <sup>+</sup> 16]
<b>CRAFT</b>	Round-based encryption without tweak	[BLMR19]
<b>PRESENT-80</b>	Nibble-serial encryption	[BKL <sup>+</sup> 07]
<b>LED-64</b>	Round-based encryption	[GPPR11]
<b>Midori-64</b>	Round-based encryption/decryption	[BBI <sup>+</sup> 15]

but has certainly a considerably-higher area overhead compared to the clock-gating implementation.

### 4.3 Outcomes

Considering the shown case studies, the following conclusions can be made.

- It should be tried to provide an optimized unprotected implementation. Here, the optimization reflects the number of 2-input non-linear gates and how they are cascaded. A lower number of such gates would naturally reduce the number of required fresh mask bits, and a lower number of cascaded ones would lead to a lower number of added latency cycles. Note that area-optimized unprotected implementations, would not necessarily make optimized masked implementations (see Table 6).

For example, Canright’s design [Can05] has a lower area footprint compared to the design of Boyar and Peralta [BP12]. Although both designs have at most 4 cascaded 2-input non-linear gates, the construction of Canright instantiates more such gates. Hence, the masked implementations of both have the same number of added latency cycles, but the Canright construction demands for a higher number of fresh mask bits (Table 6 vs. Table 7).

- A suitable choice for the processing method depends on several factors mainly related to the designer’s goal(s). For instance, if the lowest latency is desired, ANF combined with GHPC and particularly GHPC<sub>LL</sub> are the right choices, having in mind that such a low latency comes at the cost of a high area overhead and a high demand for fresh randomness while being restricted to only first-order security.

The benefit of the BDD processing methods lies on their independence to the optimality of the given unprotected implementation. Here, the amount of added latency cycles is defined by the number of primary inputs. For example, in the Skinny S-box, the primary input has 4 bits, hence 4 stages of MUXes are cascaded which results in 8 cycles of added latency when HPC2 gadgets are used. It can be seen that the number of added latency cycles shown in Tables 3-7 does not depend on the level of optimality of the unprotected implementation. There is still some differences between their required number of fresh mask bits which originates from the fact that by variable reordering in BDDs slightly different circuits are constructed.

When the number of required fresh masks is the key factor, our Naive method seems to be the best choice, given that the unprotected implementation is optimized with respect to the number of 2-input non-linear gates.

- With respect to the masking scheme, AGEMA currently supports HPC1, HPC2, GHPC and GHPC<sub>LL</sub><sup>6</sup>. Apart from the fact that the last two options just cover

<sup>6</sup>Any other customized gadget can be easily added to the library of AGEMA.

first-order security, HPC2 is preferred to HPC1 due to its lower number of required fresh mask bits. However, looking at the results shown in Tables 3-7, employing HPC1 in Naive method leads to lower area overhead, particularly for higher security orders.

- The effect of the latency overhead gets more obvious by considering the full cipher implementations since the number of added latency cycles are repeated at every evaluation cycle of the circuit. For a better understanding of this fact, we refer to Figure 1, where after application of AGEMA, the combinational part of the circuit (which is part of the sequential loop) requires the added latency cycles. Comparing the results of ANF (particularly when combined with GHPC<sub>LL</sub>) with other processing methods in full cipher implementations highlights its extraordinary benefit with respect to the full latency.

In short, we should say that apart from the above observations, the designers can try different settings and examine which one fits the best to their needs. Having the netlist of the unprotected implementation in hand, generation of the masked design by AGEMA for each setting takes a couple of seconds. As explained in Section 3.6, the resulting design together with the RTL of the gadgets should be synthesized to obtain the final performance figures (area and delay). However, the number of added latency cycles and the amount of required fresh masks (which are known right after the generation of the masked circuit by AGEMA) already give an overview of the suitability of the constructed circuit.

The tool and all case studies are provided in the GitHub: <https://github.com/Chair-for-Security-Engineering/AGEMA>.

#### 4.3.1 Comparison with Hand-Crafted Designs

As given in Section 1, manual construction of masked hardware is a time-consuming and error-prone process. Further, since such hand-crafted designs are usually not based on any composable security notion, the security of the final designs cannot be easily proven or even evaluated. It might be possible to examine the security of the constructed masked S-box using SILVER, but it is out of the feasibility bounds of such tools to examine the full cipher implementations. We can exemplarily refer to 2-share first-order masked AES S-box designs [GMK16, CRB<sup>+</sup>16], which can still be evaluated by SILVER, but not higher-order ones or those which make use of more than 2 shares [MPL<sup>+</sup>11, BGN<sup>+</sup>14, BGN<sup>+</sup>15, Sug19, GMK16, CRB<sup>+</sup>16]. Further, due to the aforementioned difficulty, to the best of our knowledge, no secure manually-crafted masked hardware design at third (or higher) order has been reported in literature. We are only aware of [GMK17] and [GSM17] which are based on DOM multiplier [GMK16], while its security at higher orders has been criticized in [MMSS19]. In contrast, the security of circuits constructed by AGEMA utilizing PINI gadgets is formally provable. More details are given in the next section.

Apart from such shortcomings and difficulties, manually-crafted designs are usually more efficient in terms of performance, i.e., area overhead, number of added latency cycles, and amount of required fresh masks. For example, several works have been published with the goal of minimizing the demand for fresh randomness in first-order masked AES S-box [BGN<sup>+</sup>14, BGN<sup>+</sup>15, Sug19, GMK16, CRB<sup>+</sup>16, SM20]. We have listed the performance figure of some manually-constructed full cipher implementations in the tables given in Appendix A.

## 5 Analyses

### 5.1 Theoretical

As explained in Section 2.4, the concept behind composable security implies that if standalone secure sub-circuits fulfill certain requirements, their composition would maintain the same level of security. Currently, PINI is known as the most efficient solution which defines such requirements. In other words, if sub-circuits, i.e., gadgets, are PINI secure under the (glitch-extended) probing model, and their interconnections do not intermix share domains, the composed circuit is also PINI secure under (glitch-extended) probing model. Note that not mixing the share domains predicates that output shares of a gadget are connected to the input shares of another gadget in the same order. For example, having  $(y^0, y^1, y^2)$  as the output shares of a gadget with 3 shares and  $(x^0, x^1, x^2)$  as the shares of an input of another gadget, the only valid interconnection is  $(x^0, x^1, x^2) = (y^0, y^1, y^2)$ .

Hence, as the first analysis step, we have examined our implementations of HPC1, HPC2, GHPC and GHPC<sub>LL</sub> gadgets with SILVER [KSM20] and verified their PINI security under glitch-extended probing model. The gadgets include NOT, 2-input AND, NAND, OR, NOR, XOR, XNOR, and 2-to-1 MUX. We made VHDL/Verilog implementation of all gadgets parametric, i.e., the security order and whether a pipeline design is desired are easily set when instantiating such modules.

Further, since AGEMA makes use of deterministic algorithms to connect gadgets together, their accordance to PINI interconnections is guaranteed. Note that fullVerif [CGLS21] has been developed to examine this. More precisely, it receives a design where the PINI gadgets are annotated, and evaluates if their interconnections are valid, i.e., comply with the aforementioned no-mixing of share domains. For the sake of completeness, we verified the designs generated by AGEMA with fullVerif, indicating the compliance of their interconnections with that of PINI.

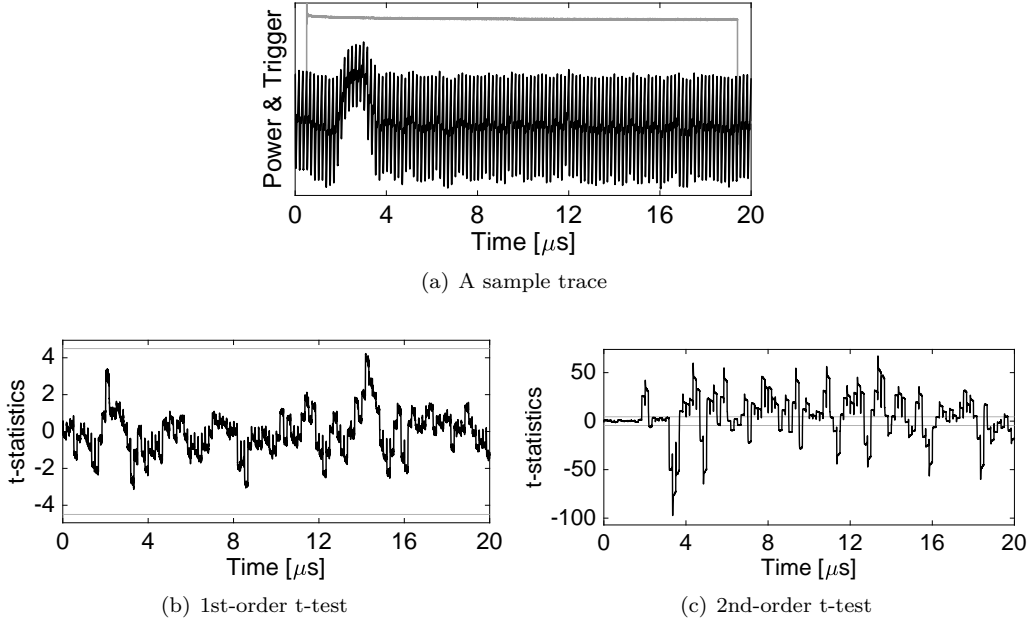
In addition to these analyses, by means of SILVER we confirmed the security of all masked Skinny S-boxes and some of the masked AES S-boxes given in Section 4.1 and listed in Tables 3-7 under the glitch-extended probing model. Note that since SILVER makes BDD of the given design to evaluate, it is restricted to small-size circuits. Therefore, we were not able to do the same for masked AES S-boxes at high security orders and also for full cipher implementations. Note that, analyzing the security of the constructed circuit using SILVER is a redundant step, since – as stated above – the security of such composed circuits is inherited from the employed PINI gadgets.

### 5.2 Experimental

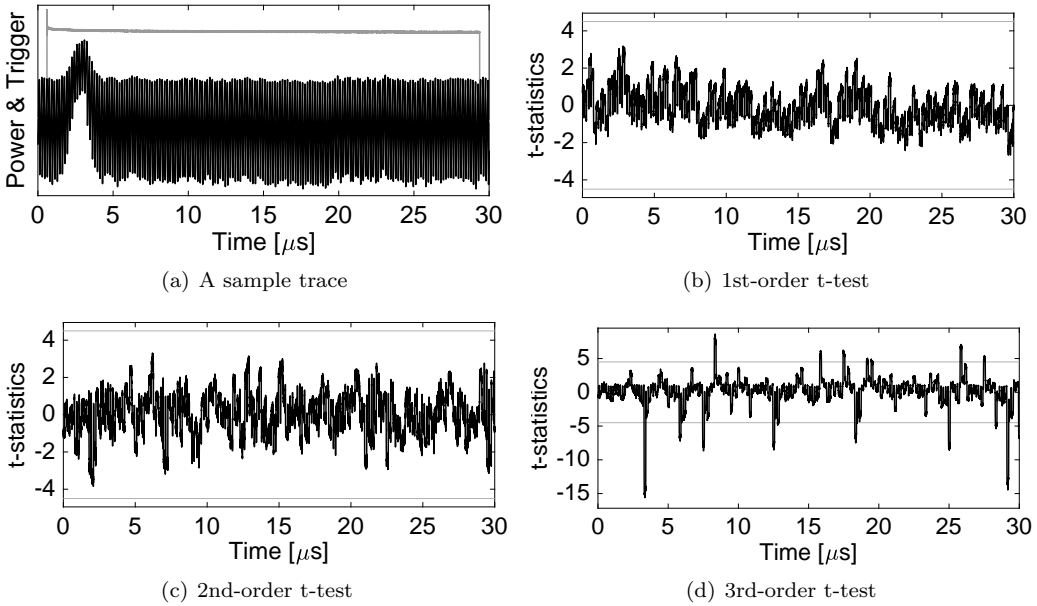
As a common evaluation technique in the state of the art and for the sake of completeness, we additionally performed Field Programmable Gate Array (FPGA)-based experimental analyses. Naturally, it is not possible to experimentally examine all designs reported as case study, and we contented ourselves with two exemplary designs of our masked Skinny round-based designs. As the first design, we selected the first-order GHPC ANF pipeline design, and as the second one the second-order HPC2 Naive pipeline design.

We made use of SAKURA-G [SAK] and implemented the selected designs on the target Spartan-6 FPGA to monitor the power consumption by a digital sampling oscilloscope at a sampling frequency of 500 MS/s. During the measurements the target design was driven by a stable 6 MHz clock. The fresh masks have also been generated internally (inside the target FPGA). For each required fresh mask bit we instantiated a 31-bit Linear Feedback Shift Register (LFSR) with the feedback polynomial  $x^{31} + x^{28} + 1$  initialized randomly<sup>7</sup>.

<sup>7</sup>We have taken the FPGA-optimized LFSR design presented in [DMW18] which requires only three instances of 6-to-1 LUTs on Xilinx FPGAs.



**Figure 9:** FPGA-based fixed versus random t-test using 100 million traces, Skinny-64-64 round-based encryption, first-order GHPC ANF pipeline.



**Figure 10:** FPGA-based fixed versus random t-test using 100 million traces, Skinny-64-64 round-based encryption, second-order HPC2 Naive pipeline.

As the analysis scheme, we conducted the common and well-known TVLA [GJJR11], where the SCA leakages associated to a fixed input are compared to those associated to random inputs. In all our experiments, we kept the key constant and performed fixed-plaintext versus random-plaintext t-tests (at first, second, and third orders). Conducting such analyses using 100 million traces (for each design) led to the results shown in Figure 9

and Figure 10 for two aforementioned designs, respectively. As the first design expected to be only first-order secure (with 2 shares,  $d = 1$ ), higher-order detected leakages are expected, as it can also be seen in the corresponding figures. The second design should be secure up to the second order (with 3 shares,  $d = 2$ ) and, as shown, no first- and no second-order leakage is detected, confirming the expected level of security.

## 6 Conclusions

In this work we introduced a comprehensive framework which we have developed for *automated generation of masked hardware* (AGEMA), allowing engineers and hardware designers of all levels of experience to easily create securely masked cryptographic hardware circuits. Based on the security and composability notion of PINI, our tool explores different processing techniques to transform any unprotected cryptographic design into a securely masked circuit using different masked gadgets as fundamental building blocks.

Demonstrating the benefits and limitations of our developed tool, we provide several case studies for well-established symmetric block ciphers, showing different performance trade-offs in terms of area overhead, latency increase, and fresh randomness demands based on our proposed transformation methodologies. Eventually, verifying the viability of our tool and the security of the resulting masked circuits, we perform practical experiments and evaluations that confirm our claims. For this, AGEMA is an important building block towards security-aware Electronic Design Automation (EDA), assisting in the automation process of creating secure ICs.

Apart from unique benefits and facilities that AGEMA offers, the intensive case studies, which we have provided in this article, highlight the importance of the employed gadgets with respect to their performance. The demands for fresh randomness and the latency of the constructed masked circuits heavily depend on the employed gadgets and their requirements. In terms of latency, GHPC<sub>LL</sub> gadgets are the only known constructions with only a single additional register stage, but they are limited to only first-order security. In contrast, HPC2 gadgets, which can arbitrarily be adjusted to any security order, add two register stages to the circuit. This might be seen as just one more clock cycle, but as shown by our case studies, the latency of the resulting masked circuit is doubled compared to that with GHPC<sub>LL</sub>. This difference is seen more clearly in implementation of ciphers which employ S-boxes with a high algebraic degree (i.e., a high number of cascaded non-linear gadgets). Naturally, more research in this area is required to fill the gap. More precisely, having HPC gadgets at arbitrary security orders with only one register stage even only for 2-input non-linear gates would greatly decrease the latency of the masked circuits and their area overheads.

## Acknowledgments

The work described in this paper has been supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and through the projects 393207943 GreenSec and 435264177 SAUBER.

## A Performance Results

**Table 3:** Synthesis results, Skinny S-box lookup-table representation.

Masking Scheme	Processing Method	Pipeline	Order	Area	Delay	Rand.	Latency
				[GE]	[ns]	[bit]	[cycle]
unprotected	–	–	0	42	0.20	0	0
GHPC <sub>LL</sub>	ANF		1	962	0.39	64	1
GHPC <sub>LL</sub>	ANF	✓	1	1093	0.39	64	1
GHPC <sub>LL</sub>	Naive		1	809	0.42	52	5
GHPC <sub>LL</sub>	Naive	✓	1	1172	0.41	52	5
GHPC	ANF		1	1270	0.53	4	2
GHPC	ANF	✓	1	1305	0.53	4	2
GHPC	Naive		1	1137	0.30	13	10
GHPC	Naive	✓	1	1870	0.29	13	10
HPC1	Naive		1	898	0.30	26	10
HPC1	Naive		2	1854	0.34	65	10
HPC1	Naive		3	3065	0.34	130	10
HPC1	Naive		4	4501	0.39	195	10
HPC1	Naive	✓	1	1488	0.29	26	10
HPC1	Naive	✓	2	2778	0.33	65	10
HPC1	Naive	✓	3	4323	0.33	130	10
HPC1	Naive	✓	4	6094	0.38	195	10
HPC2	BDD <sub>CUDD</sub>		1	1083	0.34	17	8
HPC2	BDD <sub>CUDD</sub>		2	2879	0.43	51	8
HPC2	BDD <sub>CUDD</sub>		3	5535	0.52	102	8
HPC2	BDD <sub>CUDD</sub>		4	9009	0.57	170	8
HPC2	BDD <sub>CUDD</sub>	✓	1	2306	0.34	17	8
HPC2	BDD <sub>CUDD</sub>	✓	2	5025	0.42	51	8
HPC2	BDD <sub>CUDD</sub>	✓	3	8792	0.51	102	8
HPC2	BDD <sub>CUDD</sub>	✓	4	13567	0.57	170	8
HPC2	BDD <sub>SYLVAN</sub>		1	1307	0.36	21	8
HPC2	BDD <sub>SYLVAN</sub>		2	3517	0.44	63	8
HPC2	BDD <sub>SYLVAN</sub>		3	6789	0.52	126	8
HPC2	BDD <sub>SYLVAN</sub>		4	11069	0.61	210	8
HPC2	BDD <sub>SYLVAN</sub>	✓	1	2748	0.35	21	8
HPC2	BDD <sub>SYLVAN</sub>	✓	2	6047	0.44	63	8
HPC2	BDD <sub>SYLVAN</sub>	✓	3	10650	0.51	126	8
HPC2	BDD <sub>SYLVAN</sub>	✓	4	16493	0.60	210	8
HPC2	Naive		1	847	0.35	13	10
HPC2	Naive		2	2236	0.42	39	10
HPC2	Naive		3	4287	0.47	78	10
HPC2	Naive		4	6968	0.56	130	10
HPC2	Naive	✓	1	1890	0.34	13	10
HPC2	Naive	✓	2	4055	0.41	39	10
HPC2	Naive	✓	3	7034	0.47	78	10
HPC2	Naive	✓	4	10790	0.55	130	10



**Table 4:** Synthesis results, Skinny S-box optimized representation.

Masking Scheme	Processing Method	Pipeline	Order	Area	Delay	Rand.	Latency
				[GE]	[ns]	[bit]	[cycle]
unprotected	–	–	0	40	0.20	0	0
GHPC <sub>LL</sub>	ANF		1	962	0.39	64	1
GHPC <sub>LL</sub>	ANF	✓	1	1093	0.39	64	1
GHPC <sub>LL</sub>	Naive		1	335	0.40	16	2
GHPC <sub>LL</sub>	Naive	✓	1	480	0.39	16	2
GHPC	ANF		1	1270	0.53	4	2
GHPC	ANF	✓	1	1305	0.53	4	2
GHPC	Naive		1	438	0.30	4	4
GHPC	Naive	✓	1	739	0.29	4	4
HPC1	Naive		1	365	0.26	8	4
HPC1	Naive		2	698	0.31	20	4
HPC1	Naive		3	1110	0.30	40	4
HPC1	Naive		4	1591	0.35	60	4
HPC1	Naive	✓	1	621	0.26	8	4
HPC1	Naive	✓	2	1101	0.30	20	4
HPC1	Naive	✓	3	1661	0.29	40	4
HPC1	Naive	✓	4	2289	0.34	60	4
HPC2	BDD <sub>CUDD</sub>		1	1072	0.36	17	8
HPC2	BDD <sub>CUDD</sub>		2	2862	0.43	51	8
HPC2	BDD <sub>CUDD</sub>		3	5515	0.51	102	8
HPC2	BDD <sub>CUDD</sub>		4	8979	0.59	170	8
HPC2	BDD <sub>CUDD</sub>	✓	1	2225	0.35	17	8
HPC2	BDD <sub>CUDD</sub>	✓	2	4906	0.42	51	8
HPC2	BDD <sub>CUDD</sub>	✓	3	8633	0.50	102	8
HPC2	BDD <sub>CUDD</sub>	✓	4	13366	0.59	170	8
HPC2	BDD <sub>SYLVAN</sub>		1	1072	0.36	17	8
HPC2	BDD <sub>SYLVAN</sub>		2	2862	0.42	51	8
HPC2	BDD <sub>SYLVAN</sub>		3	5515	0.51	102	8
HPC2	BDD <sub>SYLVAN</sub>		4	8979	0.60	170	8
HPC2	BDD <sub>SYLVAN</sub>	✓	1	2225	0.35	17	8
HPC2	BDD <sub>SYLVAN</sub>	✓	2	4906	0.42	51	8
HPC2	BDD <sub>SYLVAN</sub>	✓	3	8633	0.50	102	8
HPC2	BDD <sub>SYLVAN</sub>	✓	4	13366	0.59	170	8
HPC2	Naive		1	353	0.32	4	4
HPC2	Naive		2	818	0.39	12	4
HPC2	Naive		3	1489	0.44	24	4
HPC2	Naive		4	2351	0.52	40	4
HPC2	Naive	✓	1	747	0.31	4	4
HPC2	Naive	✓	2	1497	0.38	12	4
HPC2	Naive	✓	3	2498	0.43	24	4
HPC2	Naive	✓	4	3736	0.52	40	4

**Table 5:** Synthesis results, AES S-box lookup-table representation.

Masking Scheme	Processing Method	Pipeline	Order	Area	Delay	Rand.	Latency
				[GE]	[ns]	[bit]	[cycle]
unprotected	–	–	0	664	0.35	0	0
GHPC <sub>LL</sub>	ANF		1	150409	0.57	2048	1
GHPC <sub>LL</sub>	ANF	✓	1	157377	1.11	2048	1
GHPC <sub>LL</sub>	Naive		1	46476	0.50	3472	17
GHPC <sub>LL</sub>	Naive	✓	1	65708	0.49	3472	17
GHPC	ANF		1	135228	0.82	8	2
GHPC	ANF	✓	1	157808	0.92	8	2
GHPC	Naive		1	66226	0.40	868	34
GHPC	Naive	✓	1	104448	0.39	868	34
HPC1	Naive		1	50267	0.33	1736	34
HPC1	Naive		2	111763	0.37	4340	34
HPC1	Naive		3	190377	0.38	8680	34
HPC1	Naive		4	284043	0.42	13020	34
HPC1	Naive	✓	1	79087	0.32	1736	34
HPC1	Naive	✓	2	155064	0.36	4340	34
HPC1	Naive	✓	3	248114	0.36	8680	34
HPC1	Naive	✓	4	356210	0.41	13020	34
HPC2	BDD <sub>CUDD</sub>		1	24841	0.55	406	16
HPC2	BDD <sub>CUDD</sub>		2	68416	0.56	1218	16
HPC2	BDD <sub>CUDD</sub>		3	132834	2.72	2436	16
HPC2	BDD <sub>CUDD</sub>		4	216733	3.23	4060	16
HPC2	BDD <sub>CUDD</sub>	✓	1	53471	0.54	406	16
HPC2	BDD <sub>CUDD</sub>	✓	2	118318	0.55	1218	16
HPC2	BDD <sub>CUDD</sub>	✓	3	208765	0.69	2436	16
HPC2	BDD <sub>CUDD</sub>	✓	4	323122	0.78	4060	16
HPC2	BDD <sub>SYLVAN</sub>		1	25077	0.52	410	16
HPC2	BDD <sub>SYLVAN</sub>		2	69065	0.57	1230	16
HPC2	BDD <sub>SYLVAN</sub>		3	134122	2.65	2460	16
HPC2	BDD <sub>SYLVAN</sub>		4	218861	3.20	4100	16
HPC2	BDD <sub>SYLVAN</sub>	✓	1	53753	0.51	410	16
HPC2	BDD <sub>SYLVAN</sub>	✓	2	119134	0.55	1230	16
HPC2	BDD <sub>SYLVAN</sub>	✓	3	210328	0.69	2460	16
HPC2	BDD <sub>SYLVAN</sub>	✓	4	325669	0.79	4100	16
HPC2	Naive		1	46854	0.48	868	34
HPC2	Naive		2	137437	0.60	2604	34
HPC2	Naive		3	272327	0.82	5208	34
HPC2	Naive		4	449461	4.33	8680	34
HPC2	Naive	✓	1	105908	0.44	868	34
HPC2	Naive	✓	2	240470	0.58	2604	34
HPC2	Naive	✓	3	429483	0.67	5208	34
HPC2	Naive	✓	4	670365	0.76	8680	34

**Table 6:** Synthesis results, AES S-box Canright representation.

Masking Scheme	Processing Method	Pipeline	Order	Area	Delay	Rand.	Latency
				[GE]	[ns]	[bit]	[cycle]
unprotected	–	–	0	246	0.39	0	0
GHPC <sub>LL</sub>	ANF		1	150409	0.57	2048	1
GHPC <sub>LL</sub>	ANF	✓	1	157377	1.11	2048	1
GHPC <sub>LL</sub>	Naive		1	2606	0.58	160	4
GHPC <sub>LL</sub>	Naive	✓	1	3664	0.57	160	4
GHPC	ANF		1	135228	0.82	8	2
GHPC	ANF	✓	1	157808	0.92	8	2
GHPC	Naive		1	3514	0.51	40	8
GHPC	Naive	✓	1	5691	0.50	40	8
HPC1	Naive		1	2779	0.47	80	8
HPC1	Naive		2	5815	0.52	200	8
HPC1	Naive		3	9640	0.53	400	8
HPC1	Naive		4	14193	0.56	600	8
HPC1	Naive	✓	1	4527	0.46	80	8
HPC1	Naive	✓	2	8469	0.51	200	8
HPC1	Naive	✓	3	13199	0.51	400	8
HPC1	Naive	✓	4	18621	0.55	600	8
HPC2	BDD <sub>CUDD</sub>		1	25157	0.53	707	16
HPC2	BDD <sub>CUDD</sub>		2	69250	0.61	2121	16
HPC2	BDD <sub>CUDD</sub>		3	134438	0.69	4242	16
HPC2	BDD <sub>CUDD</sub>		4	219369	0.74	7070	16
HPC2	BDD <sub>CUDD</sub>	✓	1	54066	0.52	707	16
HPC2	BDD <sub>CUDD</sub>	✓	2	119631	0.60	2121	16
HPC2	BDD <sub>CUDD</sub>	✓	3	211082	0.69	4242	16
HPC2	BDD <sub>CUDD</sub>	✓	4	326842	0.78	7070	16
HPC2	BDD <sub>SYLVAN</sub>		1	43210	0.73	714	16
HPC2	BDD <sub>SYLVAN</sub>		2	119554	0.75	2142	16
HPC2	BDD <sub>SYLVAN</sub>		3	232505	2.55	4284	16
HPC2	BDD <sub>SYLVAN</sub>		4	379760	3.09	7140	16
HPC2	BDD <sub>SYLVAN</sub>	✓	1	91512	0.72	714	16
HPC2	BDD <sub>SYLVAN</sub>	✓	2	204100	0.73	2142	16
HPC2	BDD <sub>SYLVAN</sub>	✓	3	361667	0.71	4284	16
HPC2	BDD <sub>SYLVAN</sub>	✓	4	561181	0.77	7140	16
HPC2	Naive		1	2629	0.52	40	8
HPC2	Naive		2	7035	0.61	120	8
HPC2	Naive		3	13521	0.66	240	8
HPC2	Naive		4	21896	0.77	400	8
HPC2	Naive	✓	1	5765	0.51	40	8
HPC2	Naive	✓	2	12401	0.60	120	8
HPC2	Naive	✓	3	21577	0.66	240	8
HPC2	Naive	✓	4	33159	0.75	400	8

**Table 7:** Synthesis results, AES S-box optimized representation.

Masking Scheme	Processing Method	Pipeline	Order	Area	Delay	Rand.	Latency
				[GE]	[ns]	[bit]	[cycle]
unprotected	–	–	0	299	150	0	0
GHPC <sub>LL</sub>	ANF		1	150409	0.57	2048	1
GHPC <sub>LL</sub>	ANF	✓	1	157377	1.11	2048	1
GHPC <sub>LL</sub>	Naive		1	2311	1.41	136	4
GHPC <sub>LL</sub>	Naive	✓	1	3382	0.64	136	4
GHPC	ANF		1	135228	0.82	8	2
GHPC	ANF	✓	1	157808	0.92	8	2
GHPC	Naive		1	3074	1.09	34	8
GHPC	Naive	✓	1	5271	0.52	34	8
HPC1	Naive		1	2448	0.92	68	8
HPC1	Naive		2	5084	1.02	170	8
HPC1	Naive		3	8388	1.07	340	8
HPC1	Naive		4	12307	1.17	510	8
HPC1	Naive	✓	1	4263	0.40	68	8
HPC1	Naive	✓	2	7839	0.44	170	8
HPC1	Naive	✓	3	12085	0.44	340	8
HPC1	Naive	✓	4	16919	0.51	510	8
HPC2	BDD <sub>CUDD</sub>		1	25161	2.08	411	16
HPC2	BDD <sub>CUDD</sub>		2	69291	2.48	1233	16
HPC2	BDD <sub>CUDD</sub>		3	134490	2.70	2466	16
HPC2	BDD <sub>CUDD</sub>		4	219462	3.24	4110	16
HPC2	BDD <sub>CUDD</sub>	✓	1	54076	0.57	411	16
HPC2	BDD <sub>CUDD</sub>	✓	2	119704	0.63	1233	16
HPC2	BDD <sub>CUDD</sub>	✓	3	211169	0.69	2466	16
HPC2	BDD <sub>CUDD</sub>	✓	4	326936	0.77	4110	16
HPC2	BDD <sub>SYLVAN</sub>		1	25072	2.08	410	16
HPC2	BDD <sub>SYLVAN</sub>		2	69081	2.55	1230	16
HPC2	BDD <sub>SYLVAN</sub>		3	134122	2.65	2460	16
HPC2	BDD <sub>SYLVAN</sub>		4	218861	3.20	4100	16
HPC2	BDD <sub>SYLVAN</sub>	✓	1	53764	0.54	410	16
HPC2	BDD <sub>SYLVAN</sub>	✓	2	119135	0.63	1230	16
HPC2	BDD <sub>SYLVAN</sub>	✓	3	210328	0.69	2460	16
HPC2	BDD <sub>SYLVAN</sub>	✓	4	325669	0.79	4100	16
HPC2	Naive		1	2346	1.32	34	8
HPC2	Naive		2	6126	1.61	102	8
HPC2	Naive		3	11716	1.68	204	8
HPC2	Naive		4	18894	1.99	340	8
HPC2	Naive	✓	1	5339	0.51	34	8
HPC2	Naive	✓	2	11205	0.61	102	8
HPC2	Naive	✓	3	19217	0.68	204	8
HPC2	Naive	✓	4	29267	0.74	340	8

**Table 8:** Synthesis results, AES byte-serial encryption function.

Masking Scheme	Processing Method	Pipeline	Order	Area	Delay	Rand.	Latency	
							[GE]	[ns]
unprotected	–	–	0	3263	0.83	0	0	227
GHPC <sub>LL</sub>	ANF		1	143778	2.67	2048	1	454
GHPC <sub>LL</sub>	ANF	✓	1	161922	2.67	2048	1	454
GHPC <sub>LL</sub>	Naive		1	10056	2.34	136	4	1135
GHPC <sub>LL</sub>	Naive	✓	1	25656	0.91	136	4	1135
GHPC	ANF		1	146894	2.67	8	2	681
GHPC	ANF	✓	1	176509	2.83	8	2	681
GHPC	Naive		1	10818	1.73	34	8	2043
GHPC	Naive	✓	1	42078	0.91	34	8	2043
HPC2	BDD <sub>CUDD</sub>		1	33124	2.56	414	16	3859
HPC2	BDD <sub>CUDD</sub>	✓	1	120293	1.16	414	16	3859
HPC2	BDD <sub>SYLVAN</sub>		1	34173	2.64	431	16	3859
HPC2	BDD <sub>SYLVAN</sub>	✓	1	122566	0.97	431	16	3859
HPC2	Naive		1	10090	2.11	34	8	2043
HPC2	Naive		2	17649	2.66	102	8	2043
HPC2	Naive		3	27026	2.71	204	8	2043
HPC2	Naive	✓	1	42146	0.98	34	8	2043
HPC2	Naive	✓	2	65583	1.41	102	8	2043
HPC2	Naive	✓	3	91149	1.01	204	8	2043
[MPL <sup>+</sup> 11]	–		1	11114		48		266
[BGN <sup>+</sup> 14]	–		1	9102		44		246
[BGN <sup>+</sup> 15]	–		1	11221		44		246
[BGN <sup>+</sup> 15]	–		1	8119		32		246
[Sug19]	–		1	17100		0		266
[GMK16]	–		1	7600		28		216
[GMK16]	–		1	7100		18		246
[CRB <sup>+</sup> 16]	–		1	6681		54		276
[SM20]	–		1	7136	6.25	1		246
[SM20]	–		1	7707	6.25	0		246
[CBR <sup>+</sup> 15]	–		2	18600		126		276
[CRB <sup>+</sup> 16]	–		2	10449		162		276
[GMK17]	–		2	10000		54		246

**Table 9:** Synthesis results, AES round-based encryption function.

Masking Scheme	Processing Method	Pipeline	Order	Area	Delay	Rand.	Latency	
							[GE]	[ns]
unprotected	–	–	0	9906	1.85	0	0	11
GHPC <sub>LL</sub>	Naive		1	52450	2.28	2720	4	55
GHPC <sub>LL</sub>	Naive	✓	1	98448	0.84	2720	4	55
GHPC	Naive		1	67193	1.48	680	8	99
GHPC	Naive	✓	1	160080	0.83	680	8	99
HPC2	Naive		1	52597	2.04	680	8	99
HPC2	Naive		2	131631	2.39	2040	8	99
HPC2	Naive		3	246924	2.53	4080	8	99
HPC2	Naive	✓	1	161440	0.82	680	8	99
HPC2	Naive	✓	2	305274	0.89	2040	8	99
HPC2	Naive	✓	3	492077	0.93	4080	8	99

**Table 10:** Synthesis results, Skinny-64-64 round-based encryption function.

Masking Scheme	Processing Method	Pipeline	Order	Area	Delay	Rand.	Latency	
				[GE]	[ns]	[bit]	added	full
unprotected	–	–	0	1494	0.52	0	0	33
GHPC <sub>LL</sub>	ANF		1	18705	0.85	1024	1	66
GHPC <sub>LL</sub>	ANF	✓	1	18789	0.85	1024	1	66
GHPC <sub>LL</sub>	Naive		1	6817	0.48	256	2	99
GHPC <sub>LL</sub>	Naive	✓	1	12725	0.46	256	2	99
GHPC	ANF		1	22850	0.80	64	2	99
GHPC	ANF	✓	1	28850	0.80	64	2	99
GHPC	Naive		1	8260	0.46	64	4	165
GHPC	Naive	✓	1	20082	0.45	64	4	165
HPC2	BDD <sub>CUDD</sub>		1	18832	1.95	280	16	561
HPC2	BDD <sub>CUDD</sub>	✓	1	68410	0.52	280	16	561
HPC2	BDD <sub>SYLVAN</sub>		1	17969	1.96	262	16	561
HPC2	BDD <sub>SYLVAN</sub>	✓	1	66933	0.52	262	16	561
HPC2	Naive		1	6895	0.55	64	4	165
HPC2	Naive		2	15193	0.61	192	4	165
HPC2	Naive		3	26777	0.65	384	4	165
HPC2	Naive	✓	1	20210	0.53	64	4	165
HPC2	Naive	✓	2	36147	0.59	192	4	165
HPC2	Naive	✓	3	56096	0.63	384	4	165
[BJK <sup>+</sup> 16]	–	✓	1	4200	0.95	0		66
[SM21]	–	✓	2	10600	1.22	128		128

**Table 11:** Synthesis results, CRAFT round-based encryption function.

Masking Scheme	Processing Method	Pipeline	Order	Area	Delay	Rand.	Latency	
				[GE]	[ns]	[bit]	added	full
unprotected	–	–	0	1066	0.58	0	0	32
GHPC <sub>LL</sub>	ANF		1	15748	0.81	1024	1	64
GHPC <sub>LL</sub>	ANF	✓	1	17605	0.63	1024	1	64
GHPC <sub>LL</sub>	Naive		1	15568	1.01	1024	4	160
GHPC <sub>LL</sub>	Naive	✓	1	25852	0.54	1024	4	160
GHPC	ANF		1	22106	0.75	64	2	96
GHPC	ANF	✓	1	27214	0.66	64	2	96
GHPC	Naive		1	21365	0.63	256	8	288
GHPC	Naive	✓	1	41951	0.54	256	8	288
HPC2	BDD <sub>CUDD</sub>		1	14927	1.13	229	8	288
HPC2	BDD <sub>CUDD</sub>	✓	1	42451	0.55	229	8	288
HPC2	BDD <sub>SYLVAN</sub>		1	17509	1.16	272	8	288
HPC2	BDD <sub>SYLVAN</sub>	✓	1	47785	0.55	272	8	288
HPC2	Naive		1	15680	0.94	256	8	288
HPC2	Naive		2	43172	1.03	768	8	288
HPC2	Naive		3	84024	1.12	1536	8	288
HPC2	Naive	✓	1	42367	0.55	256	8	288
HPC2	Naive	✓	2	87291	0.57	768	8	288
HPC2	Naive	✓	3	148316	0.50	1536	8	288
[BLMR19]	–	✓	1	5106	4.05	0		64

**Table 12:** Synthesis results, PRESENT nibble-serial encryption function.

Masking Scheme	Processing Method	Pipeline	Order	Area	Delay	Rand.	Latency	
				[GE]	[ns]	[bit]	added	full
							[cycle]	
unprotected	–	–	0	1613	0.59	0	0	543
GHPC <sub>LL</sub>	ANF		1	4727	0.89	64	1	1086
GHPC <sub>LL</sub>	ANF	✓	1	6734	0.68	64	1	1086
GHPC <sub>LL</sub>	Naive		1	4143	1.03	16	2	1629
GHPC <sub>LL</sub>	Naive	✓	1	8061	0.59	16	2	1629
GHPC	ANF		1	5177	1.04	4	2	1629
GHPC	ANF	✓	1	8945	0.70	4	2	1629
GHPC	Naive		1	4245	0.68	4	4	2715
GHPC	Naive	✓	1	12095	0.59	4	4	2715
HPC2	BDD <sub>CUDD</sub>		1	5180	1.26	22	8	4887
HPC2	BDD <sub>CUDD</sub>	✓	1	21966	0.59	22	8	4887
HPC2	BDD <sub>SYLVAN</sub>		1	5245	1.30	23	8	4887
HPC2	BDD <sub>SYLVAN</sub>	✓	1	22064	0.59	23	8	4887
HPC2	Naive		1	4160	0.99	4	4	2715
HPC2	Naive		2	6478	1.13	12	4	2715
HPC2	Naive		3	8977	1.18	24	4	2715
HPC2	Naive	✓	1	12103	0.59	4	4	2715
HPC2	Naive	✓	2	18270	0.55	12	4	2715
HPC2	Naive	✓	3	24692	0.67	24	4	2715
[PMK <sup>+</sup> 11]	–		1	2282	4.61	0		565
[SM20]	–		1	1819	4.59	0		565
[SM21]	–		2	3800	2.04	8		666

**Table 13:** Synthesis results, LED-64 round-based encryption function.

Masking Scheme	Processing Method	Pipeline	Order	Area	Delay	Rand.	Latency	
				[GE]	[ns]	[bit]	added	full
							[cycle]	
unprotected	–	–	0	2056	1.22	0	0	33
GHPC <sub>LL</sub>	ANF		1	17382	1.84	1024	1	66
GHPC <sub>LL</sub>	ANF	✓	1	19893	1.38	1024	1	66
GHPC <sub>LL</sub>	Naive		1	7611	2.03	256	2	99
GHPC <sub>LL</sub>	Naive	✓	1	13383	1.09	256	2	99
GHPC	ANF		1	22904	1.58	64	2	99
GHPC	ANF	✓	1	27309	1.23	64	2	99
GHPC	Naive		1	9056	1.60	64	4	165
GHPC	Naive	✓	1	20615	1.08	64	4	165
HPC2	BDD <sub>CUDD</sub>		1	31416	2.97	469	16	561
HPC2	BDD <sub>CUDD</sub>	✓	1	96238	0.89	469	16	561
HPC2	BDD <sub>SYLVAN</sub>		1	38243	3.02	598	16	561
HPC2	BDD <sub>SYLVAN</sub>	✓	1	110725	0.89	598	16	561
HPC2	Naive		1	7691	1.98	64	4	165
HPC2	Naive		2	16375	2.07	192	4	165
HPC2	Naive		3	28322	2.33	384	4	165
HPC2	Naive	✓	1	20743	1.07	64	4	165
HPC2	Naive	✓	2	36890	1.14	192	4	165
HPC2	Naive	✓	3	57021	1.18	384	4	165

**Table 14:** Synthesis results, Midori-64 round-based encryption/decryption function.

Masking Scheme	Processing Method	Pipeline	Order	Area	Delay	Rand.	Latency	
				[GE]	[ns]	[bit]	added	full
							[cycle]	
unprotected	–	–	0	2035	0.97	0	0	17
GHPC <sub>LL</sub>	ANF		1	19493	1.08	1024	1	34
GHPC <sub>LL</sub>	ANF	✓	1	21986	0.94	1024	1	34
GHPC <sub>LL</sub>	Naive		1	17679	1.10	1024	4	85
GHPC <sub>LL</sub>	Naive	✓	1	32898	0.95	1024	4	85
GHPC	ANF		1	23901	1.05	64	2	51
GHPC	ANF	✓	1	30539	0.85	64	2	51
GHPC	Naive		1	23508	0.96	256	8	153
GHPC	Naive	✓	1	53893	0.95	256	8	153
HPC2	BDD <sub>CUDD</sub>		1	17162	1.29	231	8	153
HPC2	BDD <sub>CUDD</sub>	✓	1	53478	0.95	231	8	153
HPC2	BDD <sub>SYLVAN</sub>		1	21123	1.27	304	8	153
HPC2	BDD <sub>SYLVAN</sub>	✓	1	61576	0.95	304	8	153
HPC2	Naive		1	17801	1.10	256	8	153
HPC2	Naive		2	46371	1.21	768	8	153
HPC2	Naive		3	88246	1.27	1536	8	153
HPC2	Naive	✓	1	54309	0.95	256	8	153
HPC2	Naive	✓	2	105198	0.67	768	8	153
HPC2	Naive	✓	3	172179	0.69	1536	8	153
[MS16]	–	✓	1	7297	4.00	0		32
[SM20]	–	✓	1	7560	4.99	0		32
[SM21]	–	✓	2	15500	2.86	128		64

## References

- [Ake78] Sheldon B. Akers. Binary Decision Diagrams. *IEEE Trans. Computers*, 27(6):509–516, 1978.
- [Apt03] Krzysztof R. Apt. *Principles of constraint programming*. Cambridge University Press, 2003.
- [BBC<sup>+</sup>19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In *ESORICS 2019*, volume 11735 of *Lecture Notes in Computer Science*, pages 300–318. Springer, 2019.
- [BBD<sup>+</sup>15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified Proofs of Higher-Order Masking. In *EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.
- [BBD<sup>+</sup>16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong Non-Interference and Type-Directed Higher-Order Masking. In *CCS 2016*, pages 116–129. ACM, 2016.
- [BBI<sup>+</sup>15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In *ASIACRYPT 2015*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.



- [BCG<sup>+</sup>12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In *ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.
- [BDF<sup>+</sup>17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. In *EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–566, 2017.
- [BDM<sup>+</sup>20] Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. Tornado: Automatic Generation of Probing-Secure Masked Bitsliced Implementations. In *EUROCRYPT 2020*, volume 12107 of *Lecture Notes in Computer Science*, pages 311–341. Springer, 2020.
- [BGG<sup>+</sup>14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the Cost of Lazy Engineering for Masked Software Implementations. In *CARDIS 2014*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.
- [BGI<sup>+</sup>18a] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal Verification of Masked Hardware Implementations in the Presence of Glitches. In *EUROCRYPT 2018*, volume 10821 of *Lecture Notes in Computer Science*, pages 321–353. Springer, 2018.
- [BGI<sup>+</sup>18b] Roderick Bloem, Hannes Groß, Rinat Iusupov, Martin Krenn, and Stefan Mangard. Sharing Independence & Relabeling: Efficient Formal Verification of Higher-Order Masking. *IACR Cryptol. ePrint Arch.*, 2018:1031, 2018.
- [BGN<sup>+</sup>14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. A More Efficient AES Threshold Implementation. In *AFRICACRYPT 2014*, volume 8469 of *Lecture Notes in Computer Science*, pages 267–284. Springer, 2014.
- [BGN<sup>+</sup>15] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Trade-Offs for Threshold Implementations Illustrated on AES. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 34(7):1188–1200, 2015.
- [BJK<sup>+</sup>16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In *CRYPTO 2016*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.
- [BKL<sup>+</sup>07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.
- [BLMR19] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks. *IACR Trans. Symmetric Cryptol.*, 2019(1):5–45, 2019.

- [BP12] Joan Boyar and René Peralta. A Small Depth-16 Circuit for the AES S-Box. In *Information Security and Privacy Conference, SEC 2012*, volume 376 of *IFIP*, pages 287–298. Springer, 2012.
- [BRB<sup>+</sup>11] Ali Galip Bayrak, Francesco Regazzoni, Philip Brisk, François-Xavier Standaert, and Paolo Ienne. A first step towards automatic application of power analysis countermeasures. In *DAC 2011*, pages 230–235. ACM, 2011.
- [BRN<sup>+</sup>15] Ali Galip Bayrak, Francesco Regazzoni, David Novo, Philip Brisk, François-Xavier Standaert, and Paolo Ienne. Automatic Application of Power Analysis Countermeasures. *IEEE Trans. Computers*, 64(2):329–341, 2015.
- [Bry86] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.
- [Can05] David Canright. A Very Compact S-Box for AES. In Josyula R. Rao and Berk Sunar, editors, *CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.
- [CBG<sup>+</sup>17] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does Coupling Affect the Security of Masked Implementations? In *COSADE 2017*, volume 10348 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2017.
- [CBR<sup>+</sup>15] Thomas De Cnudde, Begül Bilgin, Oscar Reparaz, Ventzislav Nikov, and Svetla Nikova. Higher-order threshold implementation of the AES s-box. In *CARDIS 2015*, volume 9514 of *Lecture Notes in Computer Science*, pages 259–272. Springer, 2015.
- [CGD18] Yann Le Corre, Johann Großschädl, and Daniel Dinu. Micro-architectural Power Simulator for Leakage Assessment of Cryptographic Software on ARM Cortex-M3 Processors. In *COSADE 2018*, volume 10815 of *Lecture Notes in Computer Science*, pages 82–98. Springer, 2018.
- [CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. on Computers*, 2021.
- [CGP<sup>+</sup>12] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of Security Proofs from One Leakage Model to Another: A New Issue. In *COSADE 2012*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.
- [CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
- [CPRR13] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-Order Side Channel Security and Mask Refreshing. In *FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 2013.
- [CRB<sup>+</sup>16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with  $d+1$  Shares in Hardware. In *CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212. Springer, 2016.

- [CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Information Forensics and Security*, 15:2542–2555, 2020.
- [DMW18] Lauren De Meyer, Amir Moradi, and Felix Wegener. Spin Me Right Round Rotational Symmetry for FPGA-Specific AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):596–626, 2018.
- [DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.
- [FGP<sup>+</sup>18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.
- [GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic Low-Latency Masking in Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):1–21, 2018.
- [GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for sidechannel resistance validation. In *NIST non-invasive attack testing workshop*, 2011.
- [GM18] Hannes Groß and Stefan Mangard. A unified masking approach. *J. Cryptogr. Eng.*, 8(2):109–124, 2018.
- [GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *Theory of Implementation Security - TIS@CCS 2016*, page 3. ACM, 2016.
- [GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In *CT-RSA 2017*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017.
- [GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In *CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.
- [GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In *CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.
- [GSM17] Hannes Groß, David Schaffenrath, and Stefan Mangard. Higher-Order Side-Channel Protected Implementations of Keccak. In *DSD 2017*, pages 205–212. IEEE Computer Society, 2017.
- [HS13] Michael Hutter and Jörn-Marc Schmidt. The Temperature Side Channel and Heating Fault Attacks. In *CARDIS 2013*, volume 8419 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2013.
- [Inc] Synopsys Inc. Design compiler graphical. <https://www.synopsys.com/>.
- [ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

- [KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
- [Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
- [KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - Statistical Independence and Leakage Verification. In *ASIACRYPT 2020*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.
- [KSM22] David Knichel, Pascal Sasdrich, and Amir Moradi. Generic Hardware Private Circuits - Towards Automated Generation of Composable Secure Gadgets, 2022.
- [Mea55] George H Mealy. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5):1045–1079, 1955.
- [MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):256–292, 2019.
- [MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In *CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.
- [MPL<sup>+</sup>11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.
- [MS16] Amir Moradi and Tobias Schneider. Side-Channel Analysis Protection and Low-Latency in Action - - Case Study of PRINCE and Midori -. In *ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 517–547, 2016.
- [NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In *ICICS 2006*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.
- [PMK<sup>+</sup>11] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-Channel Resistant Crypto for Less than 2, 300 GE. *J. Cryptology*, 24(2):322–345, 2011.
- [RBN<sup>+</sup>15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In *CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.
- [SAK] SAKURA. Side-channel Attack User Reference Architecture. <http://satoh.cs.uec.ac.jp/SAKURA/index.html>.

- [SM20] Aein Rezaei Shahmirzadi and Amir Moradi. Re-Consolidating First-Order Masking Schemes - Nullifying Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):305–342, 2020.
- [SM21] Aein Rezaei Shahmirzadi and Amir Moradi. Second-Order SCA Security with almost no Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3), 2021.
- [SSB<sup>+</sup>21] Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. Rosita: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers. In *NDSS 2021*. The Internet Society, 2021.
- [Sug19] Takeshi Sugawara. 3-Share Threshold Implementation of AES S-box without Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):123–145, 2019.
- [UHA17] Rei Ueno, Naofumi Homma, and Takafumi Aoki. Toward More Efficient DPA-Resistant AES Hardware Architecture Based on Threshold Implementation. In *COSADE 2017*, Lecture Notes in Computer Science, pages 50–64. Springer, 2017.
- [Wol] Clifford Wolf. Yosys open synthesis suite. <http://www.clifford.at/yosys/>.



## **Part III**

# **Conclusions**





# Chapter 7

## Conclusions and Open Problems

**Summary.** In general, this thesis highlights the persistent threat of SCA attacks and provides advances that reduce the burden for generating SCA-resistant hardware cores. Our accomplishments rest on four pillars.

In the first part of this thesis, we highlighted the importance of continuously evaluating novel SCA attack scenarios in the context of complex and modularized systems. Here, we combined publicly available software exploits and SCA to retrieve hardware-fused key material from an iPhone 4. Knowledge of this key material allowed us to perform a device-independent passcode search. We showed that the search is arbitrarily scalable and achieves an immense speedup by performing it on multiple GPUs in parallel, rather than being tied to a single device. This underscores that software and hardware should never be analyzed in isolation, but that a thorough consideration of the interplay between the two is crucial to achieving solid protection in complex systems.

The second part of this thesis dealt with formal verification of masked hardware implementations. For this, we introduced an accurate and complete tool for formal verification of common security and composability notions in the robust  $d$ -probing model. This allows for pre-silicon verification of SCA resistance solely based on the netlist of a digital circuit. For this, we unified all security and composability notions with respect to the concept of statistical independence and leverage ROBDDs to perform the actual independence check. While our original version only covered glitches as leakage-driving physical effects, our extension enables evaluation under the combined occurrence of transitions and glitches in an iterative setting. SILVER has already proven to be immensely useful in practice and was a key enabler for the design of novel, gadget-based masking schemes that are part of this thesis.

In the third part, we introduced several novel masking schemes in the context of gadget-based masking in the robust  $d$ -probing model that all offer general applicability and allow for trade-offs between the different overhead metrics. Overhead metrics we considered cover the randomness requirements, the latency and the area footprint of a protected hardware design. While it was always a trade-off between such metrics, we were able to push boundaries for latency, area footprint and randomness requirements, individually. Our novel constructions all aim to provide an increase in flexibility with respect to fine-tuning a protected implementation towards a cost trade-off most beneficial for a specific use case.

Last but not least, this thesis provides a fully functional tool for the automated generation of masked hardware. Our tool AGEMA takes an unprotected netlist as an input and outputs a netlist that is functionally equivalent, but thoroughly masked. It is possible to choose from different preprocessing methods, adjust the security level and provide a custom library of trivially composable hardware gadgets. This lowers the burden for solid masking of hardware circuits, while it hands great flexibility to the engineer to tailor a design to the considered use case.

**Open Problems.** While this thesis offers important advances with respect to the tackled challenges, there are still plenty of opportunities for future work. Up to the publication of this thesis, there is a lack of elaborations on the practicality of SCA attacks that are similar to the one we performed in the course of this thesis, but are performed on more current models of the iPhone. Furthermore, current tools performing formal verification of security and composability notions settled in the  $d$ -probing model remain limited to evaluation of rather small circuits, especially for higher security orders. A further increase in the efficiency of such tools would hence be highly beneficial. As minimizing latency is a crucial factor in hardware design for many use cases, it would be profitable to develop order-generic hardware gadgets with only a single cycle latency that can achieve extended functionality. For example, it is expected that the construction of an order-generic 3-input AND gadget would already drastically reduce the latency of automatically generated and higher-order masked cipher designs.

Another interesting open problem is how to achieve systematic and automatable masking in the context of embedded software. While there is some relevant work on methods and tools for verification and systematic masking [MOPT12, BGR18, BDM<sup>+</sup>20, ABB<sup>+</sup>21, MPW22], masking for embedded software is still far from following a systematic and generalized methodology. We are now at a point where we have a good understanding of the physical effects in hardware and how to model them sufficiently to capture the capabilities of an SCA adversary, but the understanding for software is still rather rudimentary. This is due to the variety of microarchitectural effects that occur across the heterogeneous spectrum of microcontrollers [MPW22]. As a result, it is an interesting question to see if and how the concepts and construction established for hardware masking can be translated into the context of embedded software.

**Part IV**

**Appendix**



# Bibliography

- [ABB<sup>+</sup>21] Arnold Abromeit, Florian Bache, Leon A. Becker, Marc Gourjon, Tim Güneysu, Sabrina Jorn, Amir Moradi, Maximilian Orlt, and Falk Schellenberg. Automated Masking of Software Implementations on Industrial Microcontrollers. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, pages 1006–1011. IEEE, 2021.
- [AIS18] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private Circuits: A Modular Approach. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 427–455. Springer, 2018.
- [App12] Apple. *iOS Security Guide*, May 2012.
- [App22] Apple. Apple Platform Security, May 2022. 2022. [https://help.apple.com/pdf/security/en\\_US/apple-platform-security-guide.pdf](https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf).
- [BBC<sup>+</sup>19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In Kazue Sako, Steve A. Schneider, and Peter Y. A. Ryan, editors, *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part I*, volume 11735 of *Lecture Notes in Computer Science*, pages 300–318. Springer, 2019.
- [BBD<sup>+</sup>15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified Proofs of Higher-Order Masking. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.
- [BBD<sup>+</sup>16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong Non-Interference and Type-Directed Higher-Order Masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129, 2016.
- [BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *CHES*, pages 16–29, 2004.

- [BCP<sup>+</sup>20] Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random Probing Security: Verification, Composition, Expansion and New Constructions. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 339–368. Springer, 2020.
- [BDF<sup>+</sup>16] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. *IACR Cryptol. ePrint Arch.*, page 912, 2016.
- [BdJD22] Ondrej Burkacky, Marc de Jong, and Julia Dragon. Semiconductor companies can take different paths to capture new opportunities as demand continues to outstrip supply. 2022. <https://www.mckinsey.com/industries/semiconductors/our-insights/strategies-to-lead-in-the-semiconductor-world>.
- [BDM<sup>+</sup>20] Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. Tornado: Automatic Generation of Probing-Secure Masked Bitsliced Implementations. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 311–341. Springer, 2020.
- [BFMT16] Pierre Belgarric, Pierre-Alain Fouque, Gilles Macario-Rat, and Mehdi Tibouchi. Side-Channel Analysis of Weierstrass and Koblitz Curve ECDSA on Android Smartphones. In Kazue Sako, editor, *Topics in Cryptology - CT-RSA 2016 - The Cryptographers’ Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, volume 9610 of *Lecture Notes in Computer Science*, pages 236–252. Springer, 2016.
- [BGG<sup>+</sup>21] Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Ortl, Clara Paglialonga, and Lars Porth. Masking in fine-grained leakage models: Construction, implementation and verification. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):189–228, 2021.
- [BGI<sup>+</sup>18] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal Verification of Masked Hardware Implementations in the Presence of Glitches. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 321–353. Springer, 2018.
- [BGN<sup>+</sup>14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-Order Threshold Implementations. In Palash Sarkar and Tetsu

- Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.
- [BGR18] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight Private Circuits: Achieving Probing Security with the Least Refreshing. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, pages 343–372, 2018.
- [Bil15] Begül Bilgin. *Threshold implementations : As Countermeasure Against Higher-Order Differential Power Analysis*. PhD thesis, University of Twente, Enschede, Netherlands, 2015.
- [BJK<sup>+</sup>16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In *CRYPTO 2016*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.
- [BK21] Nicolas Bordes and Pierre Karpman. Fast Verification of Masking Schemes in Characteristic Two. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 283–312. Springer, 2021.
- [BKL<sup>+</sup>07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In *CHES*, volume 4727, pages 450–466, 2007.
- [BKP<sup>+</sup>22] Ondrej Burkacky, Ulrike Kingsbury, Andrea Pedroni, Giulietta Poltronieri, Matt Schrimper, and Brooke Weddle. Semiconductor companies are on the cusp of a decade of opportunity, but success will be predicated on the ability to attract and retain the best and brightest. 2022. <https://www.mckinsey.com/industries/semiconductors/our-insights/how-semiconductor-makers-can-turn-a-talent-challenge-into-a-competitive-advantage>.
- [BMRT22] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. Iron-Mask: Versatile Verification of Masking Security. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 142–160. IEEE, 2022.
- [Bry86] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.
- [CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling

- Attacks Without Pre-processing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 45–68. Springer, 2017.
- [CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.
- [Che52] Herman Chernoff. A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the sum of Observations. *The Annals of Mathematical Statistics*, 23(4):493 – 507, 1952.
- [CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
- [Cor18] Jean-Sébastien Coron. Formal Verification of Side-Channel Countermeasures via Elementary Circuit Transformations. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 65–82. Springer, 2018.
- [CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.
- [CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.
- [CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):136–158, 2021.
- [Dav97] Donald W. Davies. A Brief History of Cryptography. *Inf. Secur. Tech. Rep.*, 2(2):14–17, 1997.
- [DDF19] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying Leakage Models: From Probing Attacks to Noisy Leakage. *J. Cryptol.*, 32(1):151–177, 2019.
- [DFS15] Stefan Dziembowski, Sebastian Faust, and Maciej Skorski. Noisy Leakage Revisited. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and*



- Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 159–188. Springer, 2015.
- [DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.
- [EKM<sup>+</sup>08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoqCode Hopping Scheme. In David A. Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 203–220. Springer, 2008.
- [FGP<sup>+</sup>18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.
- [FKS<sup>+</sup>22] Jakob Feldtkeller, David Knichel, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. Randomness Optimization for Gadget Compositions in Higher-Order Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):188–227, 2022.
- [FSG22] Jakob Feldtkeller, Pascal Sasdrich, and Tim Güneysu. Challenges and Opportunities of Security-Aware EDA. *ACM Transactions on Embedded Computing Systems*, 2022.
- [GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis – A Generic Side-Channel Distinguisher. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES’08*, volume 5154 of *LNCS*, pages 426–442. Springer, 2008.
- [GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic Low-Latency Masking in Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):1–21, 2018.
- [GJR] Benjamin Jun Gilbert Goodwill, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. 2011.
- [GKT19] Dennis R. E. Gnad, Jonas Krautter, and Mehdi Baradaran Tahoori. Leaky Noise: New Side-Channel Attack Vectors in Mixed-Signal IoT Devices. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):305–339, 2019.
- [GM17] Hannes Groß and Stefan Mangard. Reconciling  $d+1$  Masking in Hardware and Software. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 115–136. Springer, 2017.

- [GM18] Hannes Groß and Stefan Mangard. A Unified Masking Approach. *J. Cryptogr. Eng.*, 8(2):109–124, 2018.
- [GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, page 3. ACM, 2016.
- [GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017.
- [GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.
- [HC12] Gael Hofemeier and Robert Chesebrough. Introduction to Intel AES-NI and Intel Secure Key Instructions. *Intel, White Paper*, 62, 2012.
- [Hoe12] Jesse Hoey. The Two-Way Likelihood Ratio (G) Test and Comparison to Two-Way Chi Squared Test, 2012.
- [HPN<sup>+</sup>19] Miao He, Jungmin Park, Adib Nahiyani, Apostol Vassilev, Yier Jin, and Mark Tehranipoor. RTL-PSC: Automated power side-channel leakage assessment at register-transfer level. In *2019 IEEE 37th VLSI Test Symposium (VTS)*, pages 1–6. IEEE, 2019.
- [HS17] Sorin A Huss and Oliver Stein. A novel design flow for a security-driven synthesis of side-channel hardened cryptographic modules. *Journal of Low Power Electronics and Applications*, 7(1):4, 2017.
- [ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
- [Jr.78] Sheldon B. Akers Jr. Binary Decision Diagrams. *IEEE Trans. Computers*, 27(6):509–516, 1978.
- [JRR<sup>+</sup>18] Scott Johnson, Dominic Rizzo, Parthasarathy Ranganathan, Jon McCune, and Richard Ho. Titan: enabling a transparent silicon root of trust for cloud. In *Hot Chips: A Symposium on High Performance Chips*, volume 194, 2018.

- [KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
- [KKMP09] Markus Kasper, Timo Kasper, Amir Moradi, and Christof Paar. Breaking KeeLoq in a Flash: On Extracting Keys at Lightning Speed. In Bart Preneel, editor, *Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings*, volume 5580 of *Lecture Notes in Computer Science*, pages 403–420. Springer, 2009.
- [KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.
- [KLS22] Pantea Kiaei, Zhenyuan Liu, and Patrick Schaumont. Leverage the Average: Averaged Sampling in Pre-Silicon Side-Channel Leakage Assessment. In Ioannis Savidis, Avesta Sasan, Himanshu Thapliyal, and Ronald F. DeMara, editors, *GLSVLSI '22: Great Lakes Symposium on VLSI 2022, Irvine CA USA, June 6 - 8, 2022*, pages 3–8. ACM, 2022.
- [KM22a] David Knichel and Amir Moradi. Composable Gadgets with Reused Fresh Masks – First-Order Probing-Secure Hardware Circuits with only 6 Fresh Masks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(3):114–140, 2022.
- [KM22b] David Knichel and Amir Moradi. Low-Latency Hardware Private Circuits. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1799–1812. ACM, 2022.
- [KMM20] David Knichel, Thorben Moos, and Amir Moradi. The Risk of Outsourcing: Hidden SCA Trojans in Third-Party IP-Cores Threaten Cryptographic ICs. In *IEEE European Test Symposium, ETS 2020, Tallinn, Estonia, May 25-29, 2020*, pages 1–6. IEEE, 2020.
- [KMMS22] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. Automated Generation of Masked Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):589–629, 2022.
- [Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
- [KR11] Lars R. Knudsen and Matthew Robshaw. *The Block Cipher Companion*. Information Security and Cryptography. Springer, 2011.

- [KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - Statistical Independence and Leakage Verification. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.
- [KSM22] David Knichel, Pascal Sasdrich, and Amir Moradi. Generic Hardware Private Circuits – Towards Automated Generation of Composable Secure Gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):323–344, 2022.
- [KYL<sup>+</sup>22] Pantea Kiaei, Yuan Yao, Zhenyuan Liu, Nicole Fern, Cees-Bart Breunese, Jasper Van Woudenberg, Kate Gillis, Alex Dich, Peter Grossmann, and Patrick Schaumont. Gate-Level Side-Channel Leakage Assessment with Architecture Correlation Analysis. *CoRR*, abs/2204.11972, 2022.
- [LKMM21] Oleksiy Lisovets, David Knichel, Thorben Moos, and Amir Moradi. Let’s Take it Offline: Boosting Brute-Force Attacks on iPhone’s User Authentication through SCA. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):496–519, 2021.
- [LRB22] A. V. Lakshmy, Chester Rebeiro, and Swarup Bhunia. FORTIFY: Analytical Pre-Silicon Side-Channel Characterization of Digital Designs. In *27th Asia and South Pacific Design Automation Conference, ASP-DAC 2022, Taipei, Taiwan, January 17-20, 2022*, pages 660–665. IEEE, 2022.
- [MKSM22] Nicolai Müller, David Knichel, Pascal Sasdrich, and Amir Moradi. Transitional Leakage in Theory and Practice – Unveiling Security Flaws in Masked Circuits. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):266–288, 2022.
- [MM22] Nicolai Müller and Amir Moradi. PROLEAD A Probing-Based Hardware Leakage Detection Tool. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):311–348, 2022.
- [MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):256–292, 2019.
- [Moo22] Thorben Moos. *Physical security for next generation CMOS ICs*. PhD thesis, Ruhr University Bochum, Germany, 2022.
- [MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.
- [MOPT12] Andrew Moss, Elisabeth Oswald, Dan Page, and Michael Tunstall. Compiler Assisted Masking. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 58–75. Springer, 2012.

- [MPP16] Housseem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking Cryptographic Implementations Using Deep Learning Techniques. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, volume 10076 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 2016.
- [MPW22] Ben Marshall, Dan Page, and James Webb. MIRACLE: MICRo-ArChitectural Leakage Evaluation A study of micro-architectural power leakage across many devices. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):175–220, 2022.
- [MPZ22] Maria Chiara Molteni, Jürgen Pulkus, and Vittorio Zaccaria. On robust strong-non-interferent low-latency multiplications. *IET Inf. Secur.*, 16(2):127–132, 2022.
- [MRSS18] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage Detection with the x2-Test. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):209–237, 2018.
- [MvOV01] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.
- [MZG<sup>+</sup>22] Haocheng Ma, Qizhi Zhang, Ya Gao, Jiaji He, Yiqiang Zhao, and Yier Jin. PathFinder: side channel protection through automatic leaky paths identification and obfuscation. In Rob Oshana, editor, *DAC '22: 59th ACM/IEEE Design Automation Conference, San Francisco, California, USA, July 10 - 14, 2022*, pages 79–84. ACM, 2022.
- [NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.
- [NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schl affer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptol.*, 24(2):292–321, 2011.
- [NVI18] NVIDIA. *Geforce RTX 2080 Ti*, 2018. Link [Online, access 21-March-2021].
- [OD19] Colin O’Flynn and Alex Dewar. On-Device Power Analysis Across Hardware Security Domains. Stop Hitting Yourself. *TCHES*, 2019(4):126–153, 2019.
- [OP11] David Oswald and Christof Paar. Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World — Extended Version. available online, 2011. <http://www.emsec.rub.de/research/publications/>.
- [ORP13] David Oswald, Bastian Richter, and Christof Paar. Side-Channel Attacks on the Yubikey 2 One-Time Password Generator. to appear at RAID’13, 2013.

- [Pea95] Karl Pearson. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58:240–242, 1895.
- [PP09] Christof Paar and Jan Pelzl. *Understanding cryptography: A Textbook for Students and Practitioners*. Springer Science & Business Media, 2009.
- [PPFT22] Nitin Pundir, Jungmin Park, Farimah Farahmandi, and Mark M. Tehranipoor. Power Side-Channel Leakage Assessment Framework at Register-Transfer Level. *IEEE Trans. Very Large Scale Integr. Syst.*, 30(9):1207–1218, 2022.
- [PR13] Emmanuel Prouff and Matthieu Rivain. Masking against Side-Channel Attacks: A Formal Security Proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2013.
- [PSB<sup>+</sup>18] Emmanuel Prouff, Rémi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Dumas. Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database. *IACR Cryptol. ePrint Arch.*, page 53, 2018.
- [RBN<sup>+</sup>15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.
- [RKM19] Bastian Richter, David Knichel, and Amir Moradi. A Comparison of  $\chi^2$ -Test and Mutual Information as Distinguisher for Side-Channel Analysis. In Sonia Belaïd and Tim Güneysu, editors, *Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, volume 11833 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2019.
- [RPD<sup>+</sup>18] Chethan Ramesh, Shivukumar B. Patil, Siva Nishok Dhanuskodi, George Provelengios, Sébastien Pillement, Daniel Holcomb, and Russell Tessier. FPGA Side Channel Attacks without Physical Access. In *FCCM*, pages 45–52, 2018.
- [SBHM20] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. Low-Latency Hardware Masking with Application to AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):300–326, 2020.
- [SBM21] Aein Rezaei Shahmirzadi, Dusan Bozilov, and Amir Moradi. New First-Order Secure AES Performance Records. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):304–327, 2021.
- [SGMT18] Falk Schellenberg, Dennis R. E. Gnad, Amir Moradi, and Mehdi Baradaran Tahoori. Remote Inter-Chip Power Analysis Side-Channel Attacks at Board-Level. In *Iris*

- Bahar, editor, *Proceedings of the International Conference on Computer-Aided Design, ICCAD 2018, San Diego, CA, USA, November 05-08, 2018*, page 114. ACM, 2018.
- [SGMT21] Falk Schellenberg, Dennis R. E. Gnad, Amir Moradi, and Mehdi B. Tahoori. An Inside Job: Remote Power Analysis Attacks on FPGAs. *IEEE Des. Test*, 38(3):58–66, 2021.
- [Sha49a] Claude E. Shannon. Communication Theory of Secrecy Systems. *Bell Syst. Tech. J.*, 28(4):656–715, 1949.
- [Sha49b] Claude E. Shannon. The Synthesis of Two-Terminal Switching Circuits. *Bell Syst. Tech. J.*, 28(1):59–98, 1949.
- [Sha79] Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.
- [Sho97] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.
- [SM16] Tobias Schneider and Amir Moradi. Leakage Assessment Methodology - Extended version. *J. Cryptog. Engin.*, 6(2):85–99, 2016.
- [SM21] Aein Rezaei Shahmirzadi and Amir Moradi. Re-Consolidating First-Order Masking Schemes Nullifying Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):305–342, 2021.
- [SMG16] Tobias Schneider, Amir Moradi, and Tim Güneysu. Robust and one-pass parallel computation of correlation-based attacks at arbitrary order. In *COSADE*, volume 9689, pages 199–217, 2016.
- [SRH16] Sami Saab, Pankaj Rohatgi, and Craig Hempel. Side-Channel Protections for Cryptographic Instruction Set Extensions. *IACR Cryptol. ePrint Arch.*, page 700, 2016.
- [Stu08] Student. The probable error of a mean. *Biometrika*, pages 1–25, 1908.
- [Sug19] Takeshi Sugawara. 3-Share Threshold Implementation of AES S-box without Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):123–145, 2019.
- [TAV02] K. Tiri, M. Akmal, and I. Verbauwhede. A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withstand Differential Power Analysis on Smart Cards. In *Proceedings of the 28th European Solid-State Circuits Conference – ESSCIRC’02.*, pages 403–406, 2002.
- [Tri03] Elena Trichina. Combinational Logic Design for AES SubByte Transformation on Masked Data. *IACR Cryptol. ePrint Arch.*, page 236, 2003.
- [TV04] Kris Tiri and Ingrid Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16-20 February 2004, Paris, France*, pages 246–251. IEEE Computer Society, 2004.

- [VMC19] Aurélien Vasselle, Philippe Maurine, and Maxime Cozzi. Breaking Mobile Firmware Encryption through Near-Field Side-Channel Analysis. In Chip-Hong Chang, Ulrich Rührmair, Daniel E. Holcomb, and Patrick Schaumont, editors, *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop, ASHES@CCS 2019, London, UK, November 15, 2019*, pages 23–32. ACM, 2019.
- [Wel47] B. L. Welch. The Generalization of ‘Student’s’ Problem when Several Different Population Variances are Involved. *Biometrika*, 34(1/2):28–35, 1947.
- [YTK<sup>+</sup>21] Yuan Yao, Tuna B. Tufan, Tarun Kathuria, Baris Ege, Ulkuhan Guler, and Patrick Schaumont. Pre-silicon Architecture Correlation Analysis (PACA): Identifying and Mitigating the Source of Side-channel Leakage at Gate-level. *IACR Cryptol. ePrint Arch.*, page 530, 2021.
- [ZS18] Mark Zhao and G. Edward Suh. FPGA-Based Remote Power Side-Channel Attacks. In *S&P*, pages 229–244, 2018.
- [ZSHS12] Michael Zohner, Marc Stöttinger, Sorin A. Huss, and Oliver Stein. An adaptable, modular, and autonomous side-channel vulnerability evaluator. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2012, San Francisco, CA, USA, June 3-4, 2012*, pages 43–48. IEEE Computer Society, 2012.



# List of Abbreviations

**AES** Advanced Encryption Standard

**AI** Artificial Intelligence

**ANF** Algebraic Normal Form

**AP** Application Processor

**ASIC** Application Specific Integrated Circuit

**BDD** Binary Decision Diagram

**CBC** Cipher Block Chaining

**CMOS** Complementary Metal Oxide Semiconductor

**CMS** Consolidating Masking Schemes

**COMAR** Composable Gadgets with Reused Fresh Mask

**CPA** Correlation Power Analysis

**CTR** Counter (mode of operation)

**DAG** Directed Acyclic Graph

**DOM** Domain-Oriented Masking

**DPA** Differential Power Analysis

**DRP** Dual-Rail Precharge

**EDA** Electronic Design Automation

**EM** Electro-Magnetic

**FPGA** Field Programmable Gate Array

**GHPC** Generic Hardware Private Circuit

**GHPC<sub>LL</sub>** Low-Latency Generic Hardware Private Circuit

**GID** Group Identifier

**GPU** Graphical Processing Unit

**HPC** Hardware Private Circuit

**IC** Integrated Circuit

**IND-CPA** Indistinguishability Under Chosen Plaintext Attacks

**IND-CCA** Indistinguishability Under Chosen Ciphertext Attacks

**IoT** Internet of Things

**LFSR** Linear Feedback Shift Register

**MIA** Mutual Information Analysis

**NI** Non-Interference

**OBDD** Ordered Binary Decision Diagram

**O-PINI** Output Probe-Isolating Non-Interference

**PCB** Printed Circuit Board

**PINI** Probe Isolating Non-Interference

**p.p.t.** probabilistic polynomial-time

**PRNG** Pseudo-Random Number Generator

**RBDD** Reduced Binary Decision Diagram

**ROBDD** Reduced Ordered Binary Decision Diagram

**ROM** Read Only Memory

**RTL** Register Transfer Level

**SAT** Satisfiability

**SCA** Side-Channel Analysis

**SEP** Secure Enclave Processor

**SNI** Strong Non-Interference

**SP** Substitution-Permutation

**TI** Threshold Implementation

**TVLA** Test Vector Leakage Assessment

**SoC** System on Chip

**UID** Unique Identifier



# List of Figures

1.1	CPA setup: EM probe placed on the AP of the iPhone 4 [LKMM21]. . . . .	9
2.1	Comparison unshared vs. shared observation distribution . . . . .	28
2.2	Probe extension on exemplary circuit . . . . .	32
2.3	Security reductions between formal SCA adversary models . . . . .	33
2.4	Low noise vs. high noise in the noisy leakage model . . . . .	34
2.5	Automated Masking Concept . . . . .	36
2.6	Perfect simulatability example: Probe placement on exemplary minimal circuits .	39
2.7	From BDD to Reduced and Ordered BDD of $f(X_0, X_1, X_2) = X_0X_1 \vee X_0X_2 \vee X_1$	46



# List of Tables

1.1	Runtime comparison of passcode search: on-device vs. GPU cluster. GPU = RTX 2080 TI [LKMM21]. . . . .	9
1.2	Comparison of existing PINI-composable and glitch-robust HPCs. Gadgets in <b>bold</b> were published in the course of this thesis. $d$ denotes the security order. . .	11
2.1	Example probe which is independent of the secret $X$ but propagates into $X_0$ and $X_1$ . . . . .	40
2.2	Overview of different nonlinear hardware gadgets; constructions that are part of this thesis are marked in <b>bold</b> . . . . .	42





# About the Author

Author information as of April 2023.

## Personal Data

---

<b>Name</b>	David Knichel
<b>Adress</b>	Implementation Security Group Universitätsstr. 150, ID 2/619 44801 Bochum, Germany
<b>E-Mail</b>	david.knichel@rub.de
<b>Date of birth</b>	March 21, 1992
<b>Place of birth</b>	Essen, Germany

## Education

---

Since 05/2019	<b>PhD student</b> , <i>Ruhr-Universität Bochum</i> , Faculty for Informatics
04/2016 - 10/2018	<b>M.Sc.</b> , <i>Ruhr-Universität Bochum</i> , IT Security/Information Technology
02/2012 - 02/2016	<b>B.Sc.</b> , <i>Ruhr-Universität Bochum</i> , IT Security/Information Technology

## Professional Experience

---

10/2017 - 09/2018	<b>Student Research Assistant</b> , <i>Ruhr-Universität Bochum</i> , Embedded Security Group (EMSEC), Bochum, Germany
09/2015 - 06/2017	<b>Internship &amp; Working Student</b> , <i>NXP Semiconductors</i> , Hamburg, Germany
02/2014 - 07/2015	<b>Working Student</b> , <i>escrypt GmbH</i> , Bochum, Germany

## Awards

---

2022 **Winner of the German IT Security Award '22**, endowed with 100.000€

2016 **G Data Award**, Honor for the best Bachelor graduate of the year

# Publications and Academic Activities

## Peer-Review Journal Papers

- Oleksiy Lisovets, David Knichel, Thorben Moos, and Amir Moradi. Let's Take it Offline: Boosting Brute-Force Attacks on iPhone's User Authentication through SCA. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):496–519, 2021
- David Knichel, Pascal Sasdrich, and Amir Moradi. Generic Hardware Private Circuits – Towards Automated Generation of Composable Secure Gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):323–344, 2022
- Nicolai Müller, David Knichel, Pascal Sasdrich, and Amir Moradi. Transitional Leakage in Theory and Practice – Unveiling Security Flaws in Masked Circuits. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):266–288, 2022
- David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. Automated Generation of Masked Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):589–629, 2022
- David Knichel and Amir Moradi. Composable Gadgets with Reused Fresh Masks – First-Order Probing-Secure Hardware Circuits with only 6 Fresh Masks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(3):114–140, 2022
- Jakob Feldtkeller, David Knichel, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. Randomness Optimization for Gadget Compositions in Higher-Order Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):188–227, 2022

## Peer-Reviewed Conference Proceedings

- Bastian Richter, David Knichel, and Amir Moradi. A Comparison of  $\chi^2$ -Test and Mutual Information as Distinguisher for Side-Channel Analysis. In Sonia Belaïd and Tim Güneysu, editors, *Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, volume 11833 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2019
- David Knichel, Thorben Moos, and Amir Moradi. The Risk of Outsourcing: Hidden SCA Trojans in Third-Party IP-Cores Threaten Cryptographic ICs. In *IEEE European Test Symposium, ETS 2020, Tallinn, Estonia, May 25-29, 2020*, pages 1–6. IEEE, 2020
- David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - Statistical Independence and Leakage Verification. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology -*

*ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020

- David Knichel and Amir Moradi. Low-Latency Hardware Private Circuits. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1799–1812. ACM, 2022

## Participation in Selected Conferences and Workshops

- CARDIS 2019, *Prague, Czech Republic*
- ETS 2020, *Virtual Conference*
- Asiacrypt 2020, *Virtual Conference*
- CHES 2021, *Virtual Conference*
- COSADE 2022, *Lugano, Switzerland*
- CHES 2022, *Leuven, Belgium*
- CCS 2022, *Los Angeles, USA*